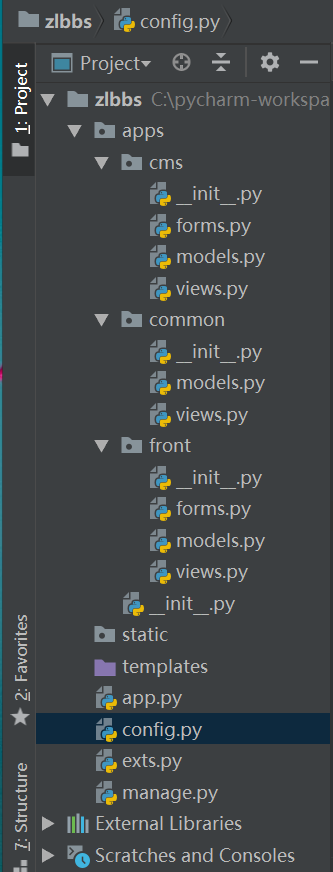
**知了课堂BBS项目开发**

# 搭建项目框架

按照以下框架搭建工程：

**zlibbs 工程总文件夹**

**apps app具体内容**

**front 前台**

\_\_init\_\_.py 初始化文件

forms.py 表单验证

models.py OMR模型

views.py 视图文件

**cms 后台**

\_\_init\_\_.py 初始化文件

forms.py 表单验证

models.py OMR模型

views.py 视图文件

**common 公共的**

\_\_init\_\_.py 初始化文件

models.py OMR模型

views.py 视图文件

config.py 配置文件

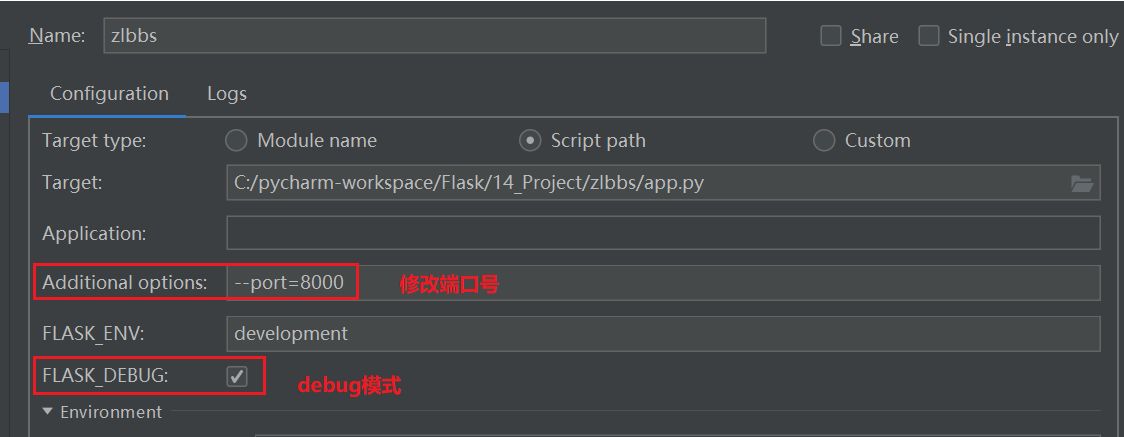
exts.py SQLAlchemy数据库，防止循环使用

manage.py 模型和数据库迁移映射文件

app.py app文件

具体看截图文件夹: 1. 搭建项目框架

打开pycharm的运行配置页面，设置debug模式和更改默认端口号为8000：



编写创建SQLAlchemy数据库对象exts.py文件，防止循环使用：

from flask\_sqlalchemy import SQLAlchemy  
  
db = SQLAlchemy()

后台视图文件：apps.cms.views.py

from flask import Blueprint  
  
bp = Blueprint('cms', \_\_name\_\_, url\_prefix='/cms')  
  
@bp.route('/')  
def index():  
 return 'cms index'

前台视图文件：apps.front.views.py

from flask import Blueprint  
  
bp = Blueprint('front', \_\_name\_\_)  
  
@bp.route('/')  
def index():  
 return 'front index'

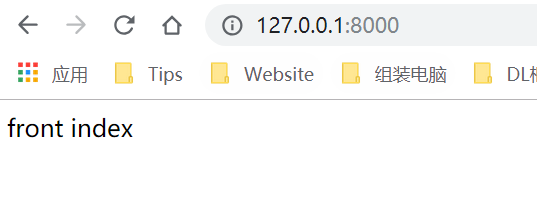
公共视图文件：apps.common.views.py

from flask import Blueprint  
  
bp = Blueprint('common', \_\_name\_\_, url\_prefix='/common')  
  
@bp.route('/')  
def index():  
 return 'common index'

在后台、前台、公共的各个包的初始化文件\_\_init\_\_.py中，加上下面这句代码：

from .views import bp

运行项目，在谷歌浏览器中打开以下链接<http://127.0.0.1:8000/>，出现以下页面：



# cms模型创建&密码加密及验证

1. 创建数据库：

**create database zlbbs charset utf8;**

1. 创建cms模型：

编写config.py文件，连接数据库：

HOSTNAME = '192.168.1.114'  
PORT = '3306'  
DATABASE = 'zlbbs'  
USERNAME = 'root'  
PASSWORD = '19881214An'  
# mysql+pymysql://root:19881214An@192.168.1.114:3306/flask\_learn?charset=utf8  
DB\_URI = "mysql+pymysql://{username}:{password}@{host}:{port}" \  
 "/{db}?charset=utf8".format(username=USERNAME, password=PASSWORD,  
 host=HOSTNAME, port=PORT, db=DATABASE)  
  
SQLALCHEMY\_DATABASE\_URI = DB\_URI  
SQLALCHEMY\_TRACK\_MODIFICATIONS = False

编写exts.py文件，防止循环使用：

from flask\_sqlalchemy import SQLAlchemy  
  
db = SQLAlchemy()

编写apps.cms.models.py文件，创建cms用户表模型：

from exts import db  
from datetime import datetime  
  
class CMSUser(db.Model):  
 \_\_tablename\_\_ = 'cms\_user'  
 id = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 username = db.Column(db.String(50), nullable=False)  
 password = db.Column(db.String(100), nullable=False)  
 email = db.Column(db.String(50), nullable=False, unique=True)  
 join\_time = db.Column(db.DateTime, default=datetime.now)

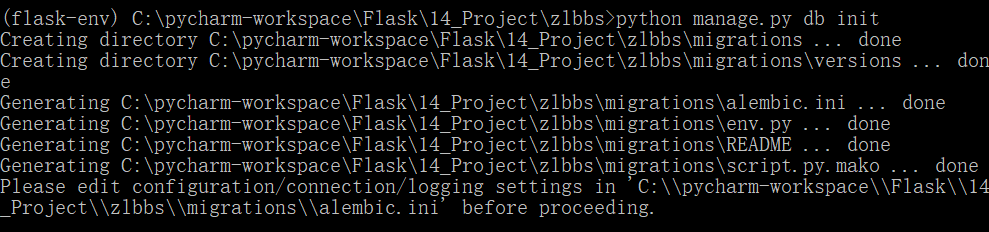
编写manage.py文件

from flask\_script import Manager  
from flask\_migrate import Migrate, MigrateCommand  
from exts import db  
from app import app  
from apps.cms import models  
  
manager = Manager()  
  
Migrate(app, db)  
manager.add\_command('db', MigrateCommand)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 manager.run()

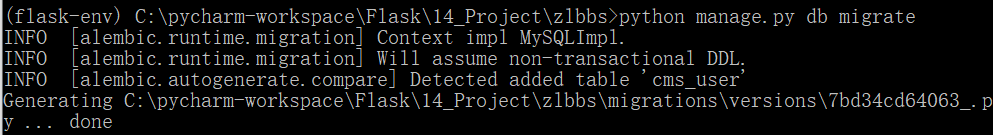
OMR模型与数据库的迁移和映射：

使用控制台，进入flask-env环境中：conda activate flask-env

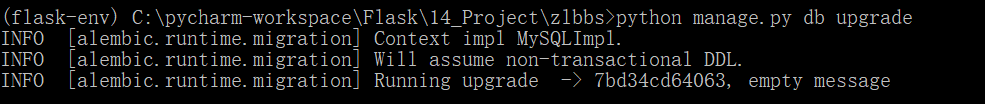
初始化迁移文件：python manage.py db init



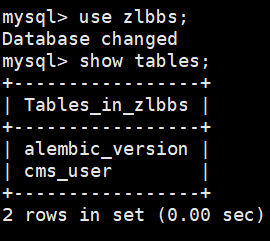
将模型的映射迁移到文件中：python manage.py db migrate



将映射文件映射到数据库中：python manage.py db upgrade



查看数据库是否映射成功，如果有cms\_user表就表示成功：



1. 添加数据到数据库表中

在manage.py文件中创建添加数据的映射函数：

@manager.option('-u', '--username', dest='username')  
@manager.option('-p', '--password', dest='password')  
@manager.option('-e', '--email', dest='email')  
def create\_cms\_user(username, password, email):  
 user = CMSUser(username=username, password=password, email=email)  
 db.session.add(user)  
 db.session.commit()  
 print('CMS用户添加成功！')

改写以下app.py文件，改变创建app为一个函数：

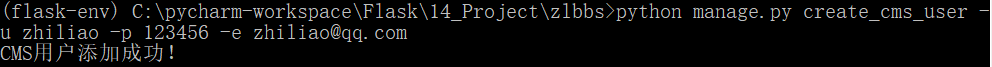
from apps.cms import bp as cms\_bp  
from apps.front import bp as front\_bp  
from apps.common import bp as common\_bp  
from exts import db  
import config  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
 app.config.from\_object(config)  
 app.register\_blueprint(cms\_bp)  
 app.register\_blueprint(front\_bp)  
 app.register\_blueprint(common\_bp)  
 db.init\_app(app)  
 return app  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = create\_app()  
 app.run()

同时更改manage.py文件，创建一个app:

from app import create\_app  
from apps.cms import models as cms\_models  
  
CMSUser = cms\_models.CMSUser  
app = create\_app()  
manager = Manager(app)  
Migrate(app, db)  
manager.add\_command('db', MigrateCommand)

在控制台，使用命令行添加一条cms用户数据：

python manage.py create\_cms\_user -u zhiliao -p 123456 -e [zhiliao@qq.com](mailto:zhiliao@qq.com)



查看数据添加结果：



1. 加密密码以及验证

修改apps.cms.models.py文件：

from exts import db  
from datetime import datetime  
from werkzeug.security import generate\_password\_hash, check\_password\_hash  
  
class CMSUser(db.Model):  
 \_\_tablename\_\_ = 'cms\_user'  
 id = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 username = db.Column(db.String(50), nullable=False)  
 # 对外字段名叫做password  
 \_password = db.Column(db.String(100), nullable=False)  
 email = db.Column(db.String(50), nullable=False, unique=True)  
 join\_time = db.Column(db.DateTime, default=datetime.now)  
  
 # 重写初始化函数  
 def \_\_init\_\_(self, username, password, email):  
 self.username = username  
 self.password = password  
 self.email = email  
  
 # 对内的字段名叫做\_password  
 @property  
 def password(self):  
 return self.\_password  
  
 # 加密密码  
 @password.setter  
 def password(self, raw\_password):  
 self.\_password = generate\_password\_hash(raw\_password)  
  
 # 验证密码  
 def check\_password(self, raw\_password):  
 result = check\_password\_hash(self.password, raw\_password)  
 return result

重新做OMR模型和数据库的映射迁移：

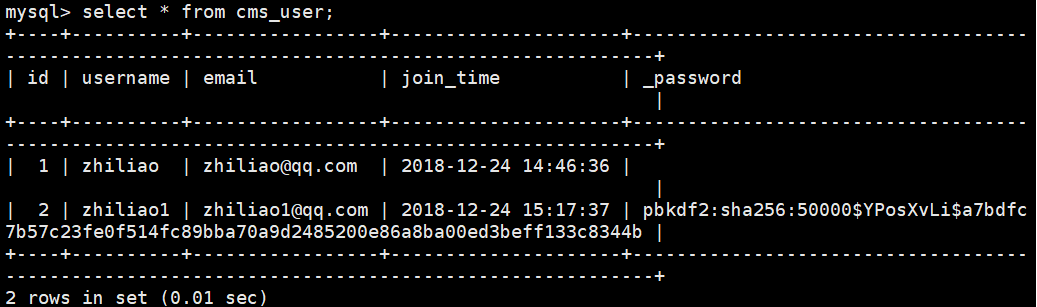
将模型的映射迁移到文件中：python manage.py db migrate

将映射文件映射到数据库中：python manage.py db upgrade

再使用控制台命令行添加一条cms用户数据：

python manage.py create\_cms\_user -u zhiliao1 -p 123456 -e [zhiliao1@qq.com](mailto:zhiliao1@qq.com)

查看cms用户数据添加结果：生成加密密码并保存到数据库表中



# cms后台登录页面完成

使用BootStrap工具来创建cms后台登录页面，链接如下：

<http://www.bootcss.com/> BootStrap中文主页

<https://v3.bootcss.com/> v3 BootStrap页面

**进入实例精选页面**

[**https://v3.bootcss.com/getting-started/#examples**](https://v3.bootcss.com/getting-started/#examples)

找到登陆页模板

<https://v3.bootcss.com/examples/signin/>

1. 编写cms后台登录页面html文件 cms\_login.html

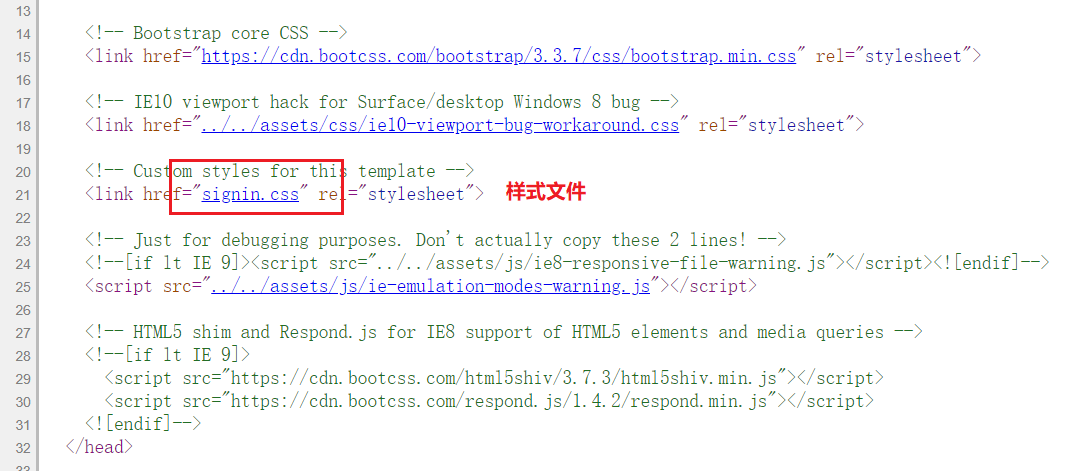
查看源代码，并且其拷贝源代码：

|  |  |
| --- | --- |
|  | <!DOCTYPE html> |
|  | <html lang="zh-CN"> |
|  | <head> |
|  | <meta charset="utf-8"> |
|  | <meta http-equiv="X-UA-Compatible" content="IE=edge"> |
|  | <meta name="viewport" content="width=device-width, initial-scale=1"> |
|  | <!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ --> |
|  | <meta name="description" content=""> |
|  | <meta name="author" content=""> |
|  | <link rel="icon" href="[../../favicon.ico](https://v3.bootcss.com/favicon.ico)"> |
|  |  |
|  | <title>Signin Template for Bootstrap</title> |
|  |  |
|  | <!-- Bootstrap core CSS --> |
|  | <link href="<https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css>" rel="stylesheet"> |
|  |  |
|  | <!-- IE10 viewport hack for Surface/desktop Windows 8 bug --> |
|  | <link href="[../../assets/css/ie10-viewport-bug-workaround.css](https://v3.bootcss.com/assets/css/ie10-viewport-bug-workaround.css)" rel="stylesheet"> |
|  |  |
|  | <!-- Custom styles for this template --> |
|  | <link href="[signin.css](https://v3.bootcss.com/examples/signin/signin.css)" rel="stylesheet"> |
|  |  |
|  | <!-- Just for debugging purposes. Don't actually copy these 2 lines! --> |
|  | <!--[if lt IE 9]><script src="../../assets/js/ie8-responsive-file-warning.js"></script><![endif]--> |
|  | <script src="[../../assets/js/ie-emulation-modes-warning.js](https://v3.bootcss.com/assets/js/ie-emulation-modes-warning.js)"></script> |
|  |  |
|  | <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries --> |
|  | <!--[if lt IE 9]> |
|  | <script src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js"></script> |
|  | <script src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script> |
|  | <![endif]--> |
|  | </head> |
|  |  |
|  | <body> |
|  |  |
|  | <div class="container"> |
|  |  |
|  | <form class="form-signin"> |
|  | <h2 class="form-signin-heading">Please sign in</h2> |
|  | <label for="inputEmail" class="sr-only">Email address</label> |
|  | <input type="email" id="inputEmail" class="form-control" placeholder="Email address" required autofocus> |
|  | <label for="inputPassword" class="sr-only">Password</label> |
|  | <input type="password" id="inputPassword" class="form-control" placeholder="Password" required> |
|  | <div class="checkbox"> |
|  | <label> |
|  | <input type="checkbox" value="remember-me"> Remember me |
|  | </label> |
|  | </div> |
|  | <button class="btn btn-lg btn-primary btn-block" type="submit">Sign in</button> |
|  | </form> |
|  |  |
|  | </div> <!-- /container --> |
|  |  |
|  |  |
|  | <!-- IE10 viewport hack for Surface/desktop Windows 8 bug --> |
|  | <script src="[../../assets/js/ie10-viewport-bug-workaround.js](https://v3.bootcss.com/assets/js/ie10-viewport-bug-workaround.js)"></script> |
|  | </body> |
|  | </html> |
|  |  |

在模板文件夹templates创建cms的登陆页面文件cms.cms\_login.html，粘贴上述源代码，并修改:

<!DOCTYPE html>  
<html lang="zh-CN">  
<head>  
 <meta charset="utf-8">  
 <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->  
 <meta name="description" content="">  
 <meta name="author" content="">  
  
 <title>登录-知了课堂论坛后台管理系统</title>  
  
 <!-- Bootstrap core CSS -->  
 <link href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
  
 <!-- Custom styles for this template -->  
 <link href="{{ url\_for('static', filename='cms/css/signin.css') }}" rel="stylesheet">  
</head>  
<body>  
<div class="container">  
 <form class="form-signin">  
 <h2 class="form-signin-heading">请登录</h2>  
 <label for="inputEmail" class="sr-only">邮箱：</label>  
 <input type="email" id="inputEmail" class="form-control" placeholder="邮箱地址" required autofocus>  
 <label for="inputPassword" class="sr-only">密码：</label>  
 <input type="password" id="inputPassword" class="form-control" placeholder="密码" required>  
 <div class="checkbox">  
 <label>  
 <input type="checkbox" value="remember-me">记住我  
 </label>  
 </div>  
 <button class="btn btn-lg btn-primary btn-block" type="submit">立即登录</button>  
 </form>  
</div> <!-- /container -->  
</body>  
</html>

在源代码中，引用了一个样式文件signin.css，如下图:



打开链接，找到其源代码，并且拷贝一份，在静态文件夹static中创建一个样式文件cms.css.signin.css，并粘贴该源代码：

body {  
 padding-top: 40px;  
 padding-bottom: 40px;  
 background-color: #eee;  
}  
  
.form-signin {  
 max-width: 330px;  
 padding: 15px;  
 margin: 0 auto;  
}  
  
.form-signin .form-signin-heading,  
.form-signin .checkbox {  
 margin-bottom: 10px;  
}  
  
.form-signin .checkbox {  
 font-weight: normal;  
}  
  
.form-signin .form-control {  
 position: relative;  
 height: auto;  
 -webkit-box-sizing: border-box;  
 -moz-box-sizing: border-box;  
 box-sizing: border-box;  
 padding: 10px;  
 font-size: 16px;  
}  
  
.form-signin .form-control:focus {  
 z-index: 2;  
}  
  
.form-signin input[type="email"] {  
 margin-bottom: -1px;  
 border-bottom-right-radius: 0;  
 border-bottom-left-radius: 0;  
}  
  
.form-signin input[type="password"] {  
 margin-bottom: 10px;  
 border-top-left-radius: 0;  
 border-top-right-radius: 0;  
}

1. 修改cms后台视图文件apps.cms.views.py：

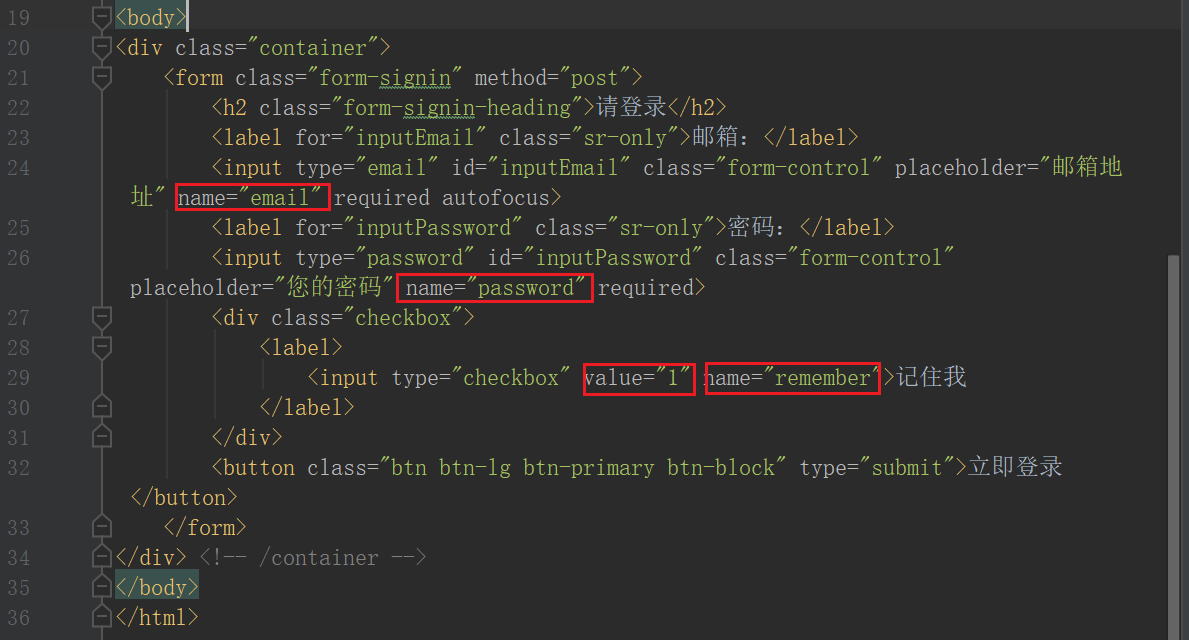
from flask import Blueprint, views, render\_template  
  
bp = Blueprint('cms', \_\_name\_\_, url\_prefix='/cms')  
  
@bp.route('/')  
def index():  
 return 'cms indexs'  
  
# 登录类视图  
class LoginView(views.MethodView):  
  
 def get(self):  
 return render\_template('cms/cms\_login.html')  
  
 def post(self):  
 pass  
  
bp.add\_url\_rule('/login/', view\_func=LoginView.as\_view('login'))

运行项目，浏览器打开链接<http://127.0.0.1:8000/cms/login/>，出现以下登录页面，说明cms后台的登录页面创建成功：



# cms后台登陆功能完成

1. 修改cms后台登录模板文件templates.cms.cms\_login.html，添加控件的name属性，如图所示：



并且修改默认的网络请求方式为POST请求：

<form class="form-signin" method="post">

1. 编写cms后台登录表单验证文件apps.cms.forms.py，验证提交的表单信息：

from wtforms import Form, StringField, IntegerField  
from wtforms.validators import Email, InputRequired, Length  
"""  
表单验证  
"""  
  
class LoginForm(Form):  
 email = StringField(validators=[Email(message='请输入正确的邮箱格式'), InputRequired(message='请输入邮箱地址')])  
 password = StringField(validators=[Length(6, 20, message='请输入正确格式的密码')])  
 remember = IntegerField()

1. 修改cms后台视图文件apps.cms.views.py，添加验证表单的功能：

from flask import Blueprint, views, render\_template, request, session, redirect, url\_for  
from .forms import LoginForm  
from .models import CMSUser  
  
bp = Blueprint('cms', \_\_name\_\_, url\_prefix='/cms')  
  
@bp.route('/')  
def index():  
 return 'cms index'  
  
# 登录类视图  
class LoginView(views.MethodView):  
  
 def get(self):  
 return render\_template('cms/cms\_login.html')  
  
 def post(self):  
 form = LoginForm(request.form)  
 if form.validate():  
 email = form.email.data  
 password = form.password.data  
 remember = form.remember.data  
  
 user = CMSUser.query.filter\_by(email=email).first()  
 if user and user.check\_password(password):  
 session['user\_id'] = user.id  
 if remember:  
 # 如果设置session.permanent = True, 那么过期时间是31天  
 session.permanent = True  
 return redirect(url\_for('cms.index'))  
 else:  
 return '邮箱或者密码错误'  
 else:  
 print(form.errors)  
 return '表单验证错误'  
  
bp.add\_url\_rule('/login/', view\_func=LoginView.as\_view('login'))

1. 在配置文件config.py中设置随机的SECRET\_KEY：

import os  
  
"""  
配置文件  
"""  
  
SECRET\_KEY = os.urandom(24) # 设置随机的SECRET\_KEY

1. 进入cmd控制台，使用命令添加一条新的cms用户数据到数据库：

python manage.py create\_cms\_user -u zhiliao2 -p 123456 -e [zhiliao2@qq.com](mailto:zhiliao2@qq.com)

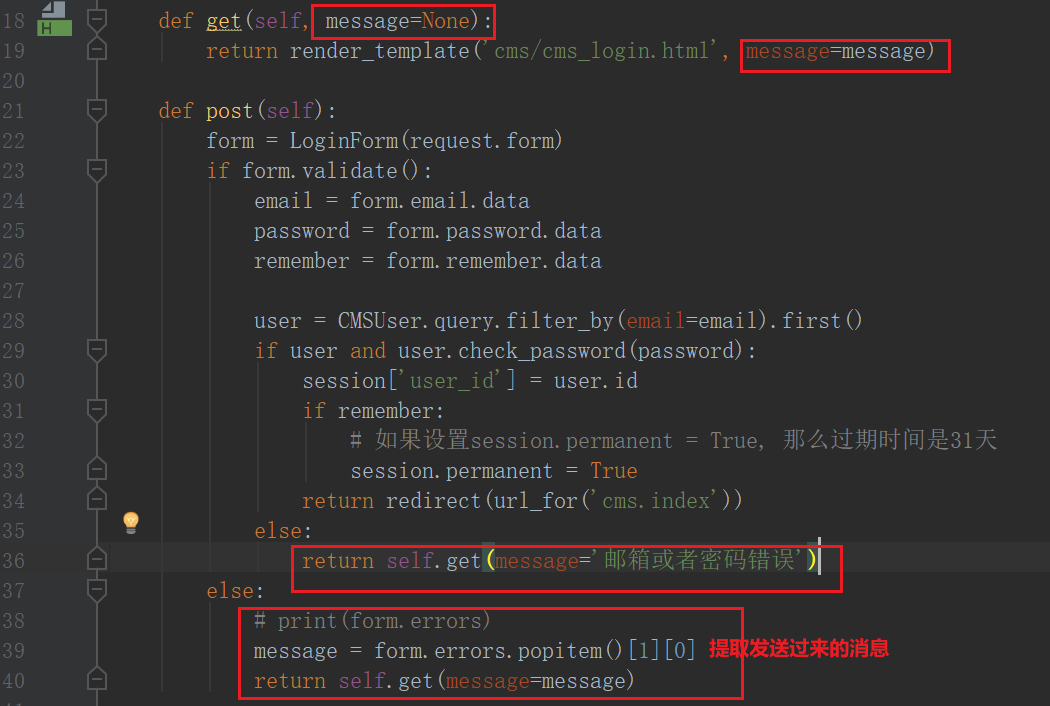
1. 启动项目，在浏览器中打开链接<http://127.0.0.1:8000/cms/login/>：

输入正确的邮箱地址和密码，不打勾记住我，页面转到前台主页面，查看session的过期时间为截至浏览会话结束；

输入正确的邮箱地址和密码，打勾记住我，页面转到前台主页面，查看session的过期时间为31天；

输入错误的邮箱地址和密码，页面提示表单验证错误，但是错误提示并不雅观。

1. 修改cms后台视图文件apps.cms.views.py，优化错误信息提示：



1. 修改cms\_login.html文件，把错误信息渲染到cms后台的登录页面上：

</button>  
 </form>  
 {# 渲染错误信息提示 #}  
 {% if message %}  
 <p style="text-align: center;" class="text-danger">{{ message }}</p>  
 {% endif %}  
</div> <!-- /container -->  
</body>  
</html>

1. 重启项目，验证错误消息样式

输入正确的邮箱，错误的密码，页面跳转回登录页面，显示错误信息，如下图：



# cms后台登陆限制

1. 在配置文件config.py中添加一个cms用户的id字串，把id固定下来：

CMS\_USER\_ID = 'ahijdnknj' # 随意设置cms用户的id字串

1. 在cms后台中，创建一个装饰器apps.cms.decorators.py文件:

from flask import session, redirect, url\_for  
from functools import wraps  
import config  
  
"""  
装饰器。  
"""  
  
# cms后台登陆限制，如果用户没有登陆，就会自动跳转到cms登陆页面  
# 只有登陆成功，才会跳转到主页面  
def login\_required(func):  
 @wraps(func)  
 def inner(\*args, \*\*kwargs):  
 if config.CMS\_USER\_ID in session:  
 return func(\*args, \*\*kwargs)  
 else:  
 return redirect(url\_for('cms.login'))  
 return inner

1. 修改cms后台视图文件views.py，把用户的id修改为配置文件config.py中固定的字串

import config

user = CMSUser.query.filter\_by(email=email).first()  
if user and user.check\_password(password):  
 session[config.CMS\_USER\_ID] = user.id # 指定用户id为固定的字串

if remember:  
 # 如果设置session.permanent = True, 那么过期时间是31天

1. 重启项目，验证结果， 在浏览器中输入http://127.0.0.1:8000/cms/：

如果用户没有登陆，页面会自动跳转到cms登陆页面<http://127.0.0.1:8000/cms/login/>;

直到登陆后，才会跳转到主页面，链接为<http://127.0.0.1:8000/cms/>。

# cms后台模板渲染完成

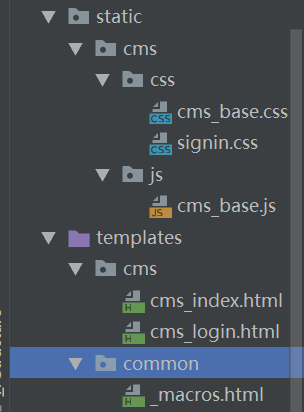
1. 创建cms后台主页模板文件cms\_index.html，把设计好的后台主页模板代码拷贝过来，并且修改指定地方:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>知了课堂后台管理系统主页</title>  
 <script src="http://cdn.bootcss.com/jquery/3.1.1/jquery.min.js"></script>  
 <link href="http://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script src="http://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
 {# 修改此处，引入css文件 #}  
 <link rel="stylesheet" href="{{ url\_for('static', filename='cms/css/cms\_base.css') }}">  
 {# 修改此处，引入js文件 #}  
 <script src="{{ url\_for('static', filename='cms/js/cms\_base.js') }}"></script>

</head>  
<body>  
<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">  
 <div class="container-fluid">

1. 拷贝设计好的静态样式和js文件

把cms\_base.css拷贝到static.cms.css文件夹中，把cms\_base.js拷贝到static.cms.js文件夹中，另外，创建一个宏文件\_macros.html，放入templates.common文件夹中，待用：



1. 修改cms后台视图文件views.py，添加主页模板渲染

@bp.route('/')  
@login\_required  
def index():  
 return render\_template('cms/cms\_index.html')

1. 重启项目， 浏览器打开登录页，输入正确的邮箱和密码，登陆，验证是否成功跳转到cms后台主页面。
2. 使用宏，优化后台主页面渲染

在宏文件\_macros.html中写入以下宏代码：

{% macro static(filename) -%}  
 {{ url\_for('static', filename=filename) }}  
{%- endmacro %}  
{# 添加减号，消除换行问题 #}

1. 在cms后台主页面模板中引入宏，并修改相应的地方：

{% from 'common/\_macros.html'**import** static %}  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>知了课堂后台管理系统主页</title>  
 <script src="http://cdn.bootcss.com/jquery/3.1.1/jquery.min.js"></script>  
 <link href="http://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script src="http://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
 {# 修改此处，引入css文件 #}  
{# <link rel="stylesheet" href="{{ url\_for('static', filename='cms/css/cms\_base.css') }}">#}  
 <link rel="stylesheet" href="{{ static('cms/css/cms\_base.css') }}">  
 {# 修改此处，引入js文件 #}  
{# <script src="{{ url\_for('static', filename='cms/js/cms\_base.js') }}"></script>#}  
 <script src="{{ static('cms/js/cms\_base.js') }}"></script>  
</head>

1. 再次启动项目，验证是否正确跳转到cms后台主页面。。。

# cms用户名渲染和注销功能完成

1. 使用上下文钩子函数before\_request，并使用线程隔离的g对象来传递变量，实现cms用户名渲染，修改cms视图文件views.py：

from flask import Blueprint, views, render\_template, request, session, redirect, url\_for, g

# 钩子函数，线程隔离的g对象，实现cms用户名渲染  
@bp.before\_request  
def before\_request():  
 if config.CMS\_USER\_ID in session:  
 user\_id = session.get(config.CMS\_USER\_ID)  
 user = CMSUser.query.get(user\_id)  
 if user:  
 g.cms\_user = user

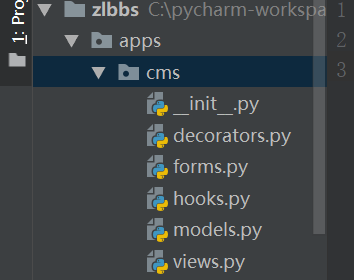
修改cms后天主页模板cms\_index.html，接收g对象传递的用户名，实现用户名渲染：

<div id="navbar" class="navbar-collapse collapse">  
 <ul class="nav navbar-nav navbar-right">  
 <li><a href="#">{{ g.cms\_user.username }}<span>[超级管理员]</span></a></li>  
 <li><a href="#">注销</a></li>  
 </ul>  
 <form class="navbar-form navbar-right">  
 <input type="text" class="form-control" placeholder="查找...">  
 </form>  
</div>

重启项目，在浏览器正确登陆，查看用户名是否显示在主页上：



优化代码， 在apps.cms文件夹中创建一个hooks.py文件，专门存放钩子函数：



把以上的钩子函数剪切到hooks.py文件上来：

from .views import bp  
from flask import session, g  
from .models import CMSUser  
import config  
  
"""  
存放钩子函数  
"""  
  
# 钩子函数，线程隔离的g对象，实现cms用户名渲染  
@bp.before\_request  
def before\_request():  
 if config.CMS\_USER\_ID in session:  
 user\_id = session.get(config.CMS\_USER\_ID)  
 user = CMSUser.query.get(user\_id)  
 if user:  
 g.cms\_user = user

在同包的初始化文件\_\_init\_\_.py上导出：

import apps.cms.hooks

重启项目，在浏览器正确登陆，查看用户名是否显示在主页上。

1. 注销功能实现

修改cms视图文件views.py，添加注销功能函数：

# 注销视图函数  
@bp.route('/logout/')  
@login\_required  
def logout():  
 del session[config.CMS\_USER\_ID] # 删除session中的用户id字串  
 return redirect(url\_for('cms.login')) # 返回登陆页面

修改cms后台主页模板，接收注销功能：

<ul class="nav navbar-nav navbar-right">  
 <li><a href="#">{{ g.cms\_user.username }}<span>[超级管理员]</span></a></li>  
 <li><a href="{{ url\_for('cms.logout') }}">注销</a></li>  
</ul>

验证，重启项目，在浏览器正确登陆，点击后台主页上的注销，看是否成功跳转到登陆页面。

# cms模板抽离和个人详情页面完成

1. 复制原来的cms\_index.html，创建cms后台父本的页面模板，在子模版中需要更改的地方设置block模块：

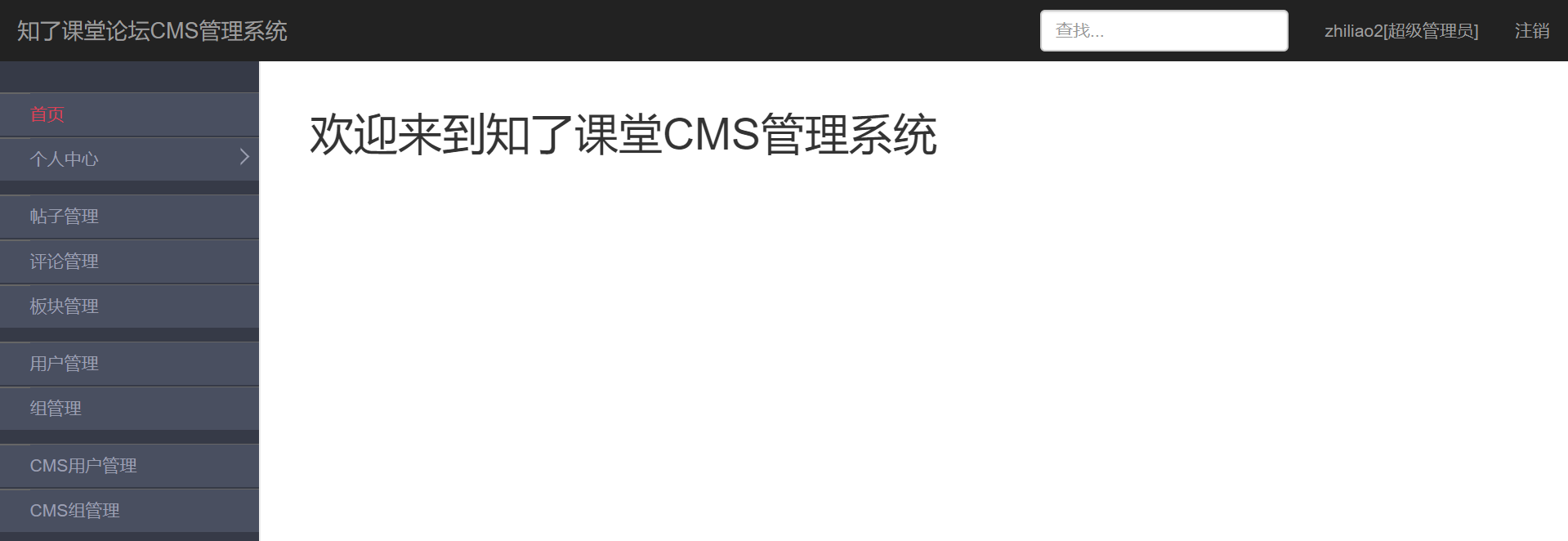
<head>  
 <meta charset="UTF-8">  
 <title>{% block title %}{% endblock %}</title> 添加block  
 <script src="http://cdn.bootcss.com/jquery/3.1.1/jquery.min.js"></script>  
 <link href="http://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script src="http://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
 {# 修改此处，引入css文件 #}  
{# <link rel="stylesheet" href="{{ url\_for('static', filename='cms/css/cms\_base.css') }}">#}  
 <link rel="stylesheet" href="{{ static('cms/css/cms\_base.css') }}">  
 {# 修改此处，引入js文件 #}  
{# <script src="{{ url\_for('static', filename='cms/js/cms\_base.js') }}"></script>#}  
 <script src="{{ static('cms/js/cms\_base.js') }}"></script>  
 {% block head %}{% endblock %} 添加block  
</head>

<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">  
 <h1>{% block page\_title %}{% endblock %}</h1> 添加block  
 <div class="main\_content">  
 {% block main\_content %}{% endblock %} 添加block  
 </div>  
</div>

1. 修改cms主页面，继承父模板，添加需要修改的内容

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}  
 知了课堂CMS管理系统  
{% endblock %}  
  
{% block page\_title %}  
 欢迎来到知了课堂CMS管理系统  
{% endblock %}

启动项目验证，登陆后，主页是以下这样的：



1. 继承父模板，实现个人详情页面：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}  
 个人信息  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title() }}   
{% endblock %}

在cms视图文件views.py中添加个人详情视图函数：

# 个人详情页面视图函数  
@bp.route('/profile/')  
@login\_required  
def profile():  
 return render\_template('cms/cms\_profile.html')

在父模板cms\_base.html中渲染个人详情：

<li class="unfold"><a href="#">首页</a></li>  
<li class="profile-li">  
 <a href="#">个人中心<span></span></a>  
 <ul class="subnav">  
 <li><a href="{{ url\_for('cms.profile') }}">个人信息</a></li> 渲染个人详情  
 <li><a href="#">修改密码</a></li>  
 <li><a href="#">修改邮箱</a></li>  
 </ul>  
</li>

启动项目验证，登陆后，主页是以下这样的：



使用BootStrap带边框的表格模板实现详情列表：

{% block main\_content %}  
 {# 设置别名 #}  
 {% set user=g.cms\_user %}  
 <table class="table table-bordered">  
 <tbody>  
 <tr>  
 <td>用户名：</td>  
 <td>{{ user.username }}</td>  
 </tr>  
 <tr>  
 <td>邮箱：</td>  
 <td>{{ user.email }}</td>  
 </tr>  
 <tr>  
 <td>角色：</td>  
 <td>暂时未实现</td>  
 </tr>  
 <tr>  
 <td>权限：</td>  
 <td>暂时未实现</td>  
 </tr>  
 <tr>  
 <td>加入时间：</td>  
 <td>{{ user.join\_time }}</td>  
 </tr>  
 </tbody>  
 </table>  
{% endblock %}

启动项目验证，登陆后，主页是以下这样的：



# cms登陆页面CSRF保护

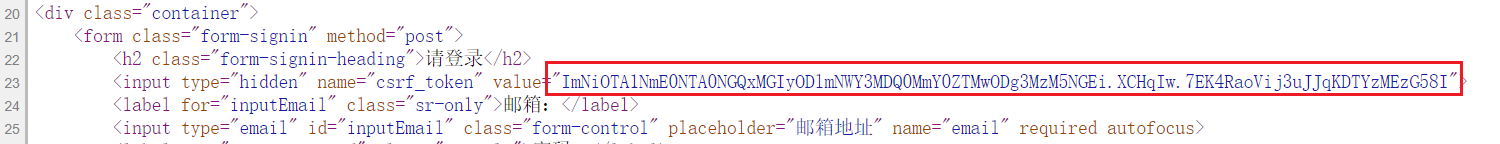
1. 项目主功能文件app.py中添加CSRF

from exts import db  
from flask\_wtf import CSRFProtect  
import config  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
 app.config.from\_object(config)  
  
 app.register\_blueprint(cms\_bp)  
 app.register\_blueprint(front\_bp)  
 app.register\_blueprint(common\_bp)  
  
 db.init\_app(app)  
 CSRFProtect(app) # 添加CSRF（跨站域请求伪造cross site request forgery）保护，防止漏洞  
  
 return app

1. 在cms登陆页面cms\_login.html中添加csrf\_token()

<div class="container">  
 <form class="form-signin" method="post">  
 <h2 class="form-signin-heading">请登录</h2>  
 <input type="hidden" name="csrf\_token" value="{{ csrf\_token() }}">  
 <label for="inputEmail" class="sr-only">邮箱：</label>

重启项目，打开cms登陆页面源代码，出现以下字串，并且能登陆成功，说明完成CSRF保护：



# cms后台修改密码页面布局完成

1. 继承父模板，创建后台修改密码页面模板cms\_resetpwd.html

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}  
 修改密码  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title() }}  
{% endblock %}  
  
{% block main\_content %}  
 这是修改密码页面  
{% endblock %}

1. 在cms视图文件上添加修改密码类视图

# 修改密码类视图  
class ResetPwdView(views.MethodView):  
 decorators = [login\_required] # 保证用户登陆  
  
 def get(self):  
 return render\_template('cms/cms\_resetpwd.html')  
  
 def post(self):  
 pass  
  
bp.add\_url\_rule('/login/', view\_func=LoginView.as\_view('login'))  
bp.add\_url\_rule('/resetpwd/', view\_func=ResetPwdView.as\_view('resetpwd'))

1. 在cms父模板中渲染修改密码页面

<li class="profile-li">  
 <a href="#">个人中心<span></span></a>  
 <ul class="subnav">  
 <li><a href="{{ url\_for('cms.profile') }}">个人信息</a></li>  
 <li><a href="{{ url\_for('cms.resetpwd') }}">修改密码</a></li> 渲染修改密码页面  
 <li><a href="#">修改邮箱</a></li>  
 </ul>  
</li>

重启项目，然后登陆，点击修改密码，出现以下页面：



1. 使用BootStrap表单，构建修改密码表单

{% block main\_content %}  
 <form action="" method="post">  
 <div class="form-container">  
 <div class="form-group">  
 <div class="input-group">  
 <span class="input-group-addon">旧密码</span>  
 <input type="password" class="form-control" name="oldpwd" placeholder="请输入旧密码">  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="input-group">  
 <span class="input-group-addon">新密码</span>  
 <input type="password" class="form-control" name="newpwd" placeholder="请输入新密码">  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="input-group">  
 <span class="input-group-addon">确认密码</span>  
 <input type="password" class="form-control" name="newpwd2" placeholder="请输入确认密码">  
 </div>  
 </div>  
 <div class="form-group">  
 <button class="btn btn-primary">立即保存</button>  
 </div>  
 </div>  
 </form>  
{% endblock %}

给这个form-container添加一个样式：

# form-container样式 #}  
{% block head %}  
 <style>  
 .form-container{  
 width: 300px;  
 }  
 </style>  
{% endblock %}

1. 重启项目，然后登陆，点击修改密码，出现以下页面：

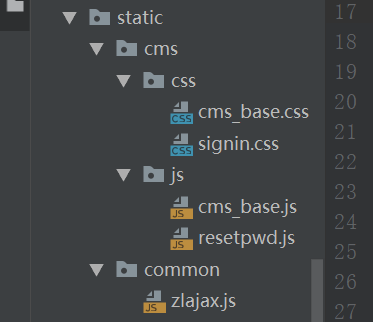


# cms后台修改密码ajax功能完成

1. 首先给cms后台修改密码的确认按钮设置一个id：



1. 在静态static.cms.js文件夹中创建一个修改密码的resetpwd.js，并且把对jquery的ajax的封装工具zlajax.js拷贝到static.common中，实现ajax处理csrf漏洞：



以下是对jquery的ajax的封装工具zlajax.js的源代码：

'use strict';  
**var** zlajax = {  
 'get': **function** (args) {  
 args['method'] = 'get';  
 **this**.ajax(args);  
 },  
 'post': **function** (args) {  
 args['method'] = 'post';  
 **this**.ajax(args);  
 },  
 'ajax': **function** (args) {  
 // 设置csrftoken  
 **this**.\_ajaxSetup();  
 $.ajax(args);  
 },  
 '\_ajaxSetup': **function** () {  
 $.ajaxSetup({  
 'beforeSend': **function** (xhr, settings) {  
 **if** (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !**this**.crossDomain) {  
 **var** csrftoken = $('meta[name=csrf-token]').attr('content');  
 xhr.setRequestHeader("X-CSRFToken", csrftoken)  
 }  
 }  
 });  
 }  
};

实现ajax功能的resetpwd.js代码：

$(**function** () {  
 $("#submit").click(**function** (event) {  
 // 阻止按钮默认提交表单的事件  
 event.preventDefault();  
  
 // 获取输入值  
 **var** oldpwdE = $("input[name=oldpwd]");  
 **var** newpwdE = $("input[name=newpwd]");  
 **var** newpwd2E = $("input[name=newpwd2]");  
  
 **var** oldpwd = oldpwdE.val();  
 **var** newpwd = newpwdE.val();  
 **var** newpwd2 = newpwd2E.val();  
  
 // AJAX处理CSRF漏洞步骤：  
 // 1. 要在模板meta标签中渲染一个csrf\_token()  
 // 2. 在ajax请求的头部中设置X-CSRFtoken  
 zlajax.post({  
 'url': '/cms/resetpwd/',  
 'data': {  
 'oldpwd': oldpwd,  
 'newpwd': newpwd,  
 'newpwd2': newpwd2  
 },  
 'success': **function** (data) {  
 console.log(data);  
 },  
 'fail': **function** (error) {  
 console.log(error);  
 }  
 });  
 });  
});

1. 在cms后台父模板templates.cms.cms\_base.html中的meta标签中渲染一个csrf\_token()，并且导入zlajax.js:

<head>  
 <meta charset="UTF-8">  
 {# 渲染一个csrf\_token()， AJAX处理CSRF漏洞 #}  
 <meta name="csrf-token" content="{{ csrf\_token() }}">  
 <title>{% block title %}{% endblock %}</title>  
 <script src="http://cdn.bootcss.com/jquery/3.1.1/jquery.min.js"></script>  
 <link href="http://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script src="http://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
 {# 修改此处，引入css文件 #}  
 <link rel="stylesheet" href="{{ static('cms/css/cms\_base.css') }}">  
 {# 修改此处，引入js文件 #}  
 <script src="{{ static('cms/js/cms\_base.js') }}"></script>  
 {# 导入zlajax.js #}  
 <script src="{{ url\_for('static', filename='common/zlajax.js') }}"></script>   
 {% block head %}{% endblock %}  
</head>

1. 在cms后台表单验证文件apps.cms.forms.py中添加修改密码的表单验证类：

from wtforms.validators import Email, InputRequired, Length, EqualTo  
…  
# 修改密码表单验证  
class ResetpwdForm(Form):  
 oldpwd = StringField(validators=[Length(6, 20, message='请输入正确格式的旧密码')])  
 newpwd = StringField(validators=[Length(6, 20, message='请输入正确格式的新密码')])  
 newpwd2 = StringField(validators=[EqualTo('newpwd', message='确认密码必须和新密码一致')])

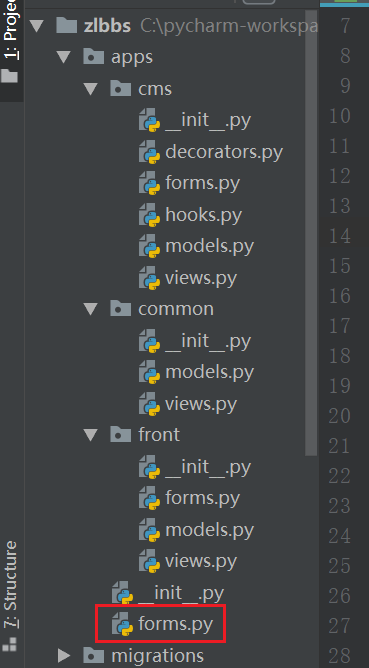
1. 在cms后台视图文件中实现修改密码验证。。。

from .forms import LoginForm, ResetpwdForm 导入修改密码表单验证类  
…

# 修改密码类视图  
class ResetPwdView(views.MethodView):  
 decorators = [login\_required] # 保证用户登陆  
  
 def get(self):  
 return render\_template('cms/cms\_resetpwd.html')  
  
 def post(self):  
 form = ResetpwdForm(request.form) # 验证修改密码  
 if form.validate():  
 pass  
 else:  
 pass

# cms后台修改密码服务逻辑完成

1. 在apps包中创建一个父本的表单验证类apps.forms.py，写入获取错误信息的方法：



apps.forms.py中的代码：

from wtforms import Form  
  
# 基础的表单验证类  
class BaseForm(Form):  
 def get\_error(self):  
 message = self.errors.popitem()[1][0] # 需要返回什么信息视具体情况而定  
 return message

1. 把cms后台的所有表单验证类都修改为基层以上的BaseForm父类：

from wtforms import Form, StringField, IntegerField  
from wtforms.validators import Email, InputRequired, Length, EqualTo  
from apps.forms import BaseForm # 导入和继承基础表单验证类  
  
"""  
表单验证  
"""  
  
# 登陆表单验证  
class LoginForm(BaseForm):  
 email = StringField(validators=[Email(message='请输入正确的邮箱'), InputRequired(message='请输入邮箱地址')])  
 password = StringField(validators=[Length(6, 20, message='请输入正确的密码')])  
 remember = IntegerField()  
  
# 修改密码表单验证  
class ResetpwdForm(BaseForm):  
 oldpwd = StringField(validators=[Length(6, 20, message='请输入正确格式的旧密码')])  
 newpwd = StringField(validators=[Length(6, 20, message='请输入正确格式的新密码')])  
 newpwd2 = StringField(validators=[EqualTo('newpwd', message='确认密码必须和新密码一致')])

这样，cms后台视图文件views.py中就可以直接是用get\_error()方法类获取验证错误的提示消息，以下是修改登陆视图中验证表单的获取错误消息的代码：

else:  
 # print(form.errors)  
 # message = form.errors.popitem()[1][0] # 提取表单验证forms.py传递过来的消息  
 message = form.get\_error() # 修改为使用父类的get\_error()方法  
 return self.get(message=message)

1. 在cms后台视图文件views.py中，完成修改密码的服务逻辑代码，返回json数据：

from flask import jsonify

from exts import db

…

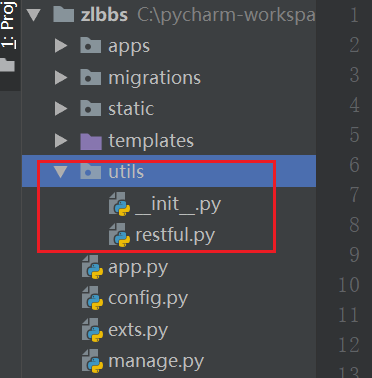
def post(self):  
 form = ResetpwdForm(request.form) # 验证修改密码  
 if form.validate():  
 oldpwd = form.oldpwd.data  
 newpwd = form.newpwd.data  
 user = g.cms\_user # 使用g对象传递用户对象  
 if user.check\_password(oldpwd):  
 # 更新数据库中的密码  
 user.password = newpwd  
 db.session.commit()  
 # 返回json数据  
 return jsonify({"code": 200, "message": ""})  
 else:  
 return jsonify({"code": 400, "message": "旧密码输入错误"})  
 else:  
 message = form.get\_error() # 使用父类的get\_error()方法获取错误信息  
 return jsonify({"code": 400, "message": message})

1. 启动项目服务，浏览器中打开登陆页面，输入正确的邮箱和密码登陆，进入修改密码页面，输入正确的旧密码和新密码，控制台出现以下消息就成功了：



# cms后台修改密码优化json数据的返回

1. 创建一个规范http状态码和返回json数据的文件utils.restful.py:



utils.restful.py代码：

from flask import jsonify  
  
"""  
前端和后端通讯的规范，优化http状态码和返回的json数据  
"""  
  
class HttpCode(object):  
 ok = 200  
 unautherror = 401  
 paramserror = 400  
 servererror = 500  
  
def restful\_result(code, message, data):  
 return jsonify({'code': code, 'message': message, 'data': data})  
  
def success(message="", data=None):  
 return restful\_result(code=HttpCode.ok, message=message, data=data)  
  
def unauth\_error(message="", data=None):  
 return restful\_result(code=HttpCode.unautherror, message=message, data=data)  
  
def params\_error(message="", data=None):  
 return restful\_result(code=HttpCode.paramserror, message=message, data=data)  
  
def server\_error(message="", data=None):  
 return restful\_result(code=HttpCode.servererror, message=message or '服务器内部错误', data=data)

1. 修改cms后台视图文件所有返回错误消息的json数据的代码：

from utils import restful

def post(self):  
 form = ResetpwdForm(request.form) # 验证修改密码  
 if form.validate():  
 oldpwd = form.oldpwd.data  
 newpwd = form.newpwd.data  
 user = g.cms\_user # 使用g对象传递用户对象  
 if user.check\_password(oldpwd):  
 # 更新数据库中的密码  
 user.password = newpwd  
 db.session.commit()  
 # 返回json数据  
 # return jsonify({"code": 200, "message": ""})  
 return restful.success()  
 else:  
 # return jsonify({"code": 400, "message": "旧密码输入错误"})  
 return restful.params\_error("旧密码错误")  
 else:  
 message = form.get\_error() # 使用父类的get\_error()方法获取错误信息  
 # return jsonify({"code": 400, "message": message})  
 return restful.params\_error(message)

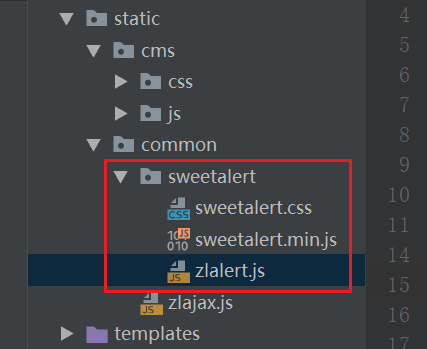
1. 启动项目服务，浏览器中打开登陆页面，输入正确的邮箱和密码登陆，进入修改密码页面，输入正确的旧密码和新密码，控制台出现以下消息就成功了：



# sweetalert优化修改密码结果反馈完成

1. 把sweetalert工具包拷贝到静态static.common中，并修改为zlalert.js名称：

这是一个已经写好的提示框工具包，直接使用就好了。



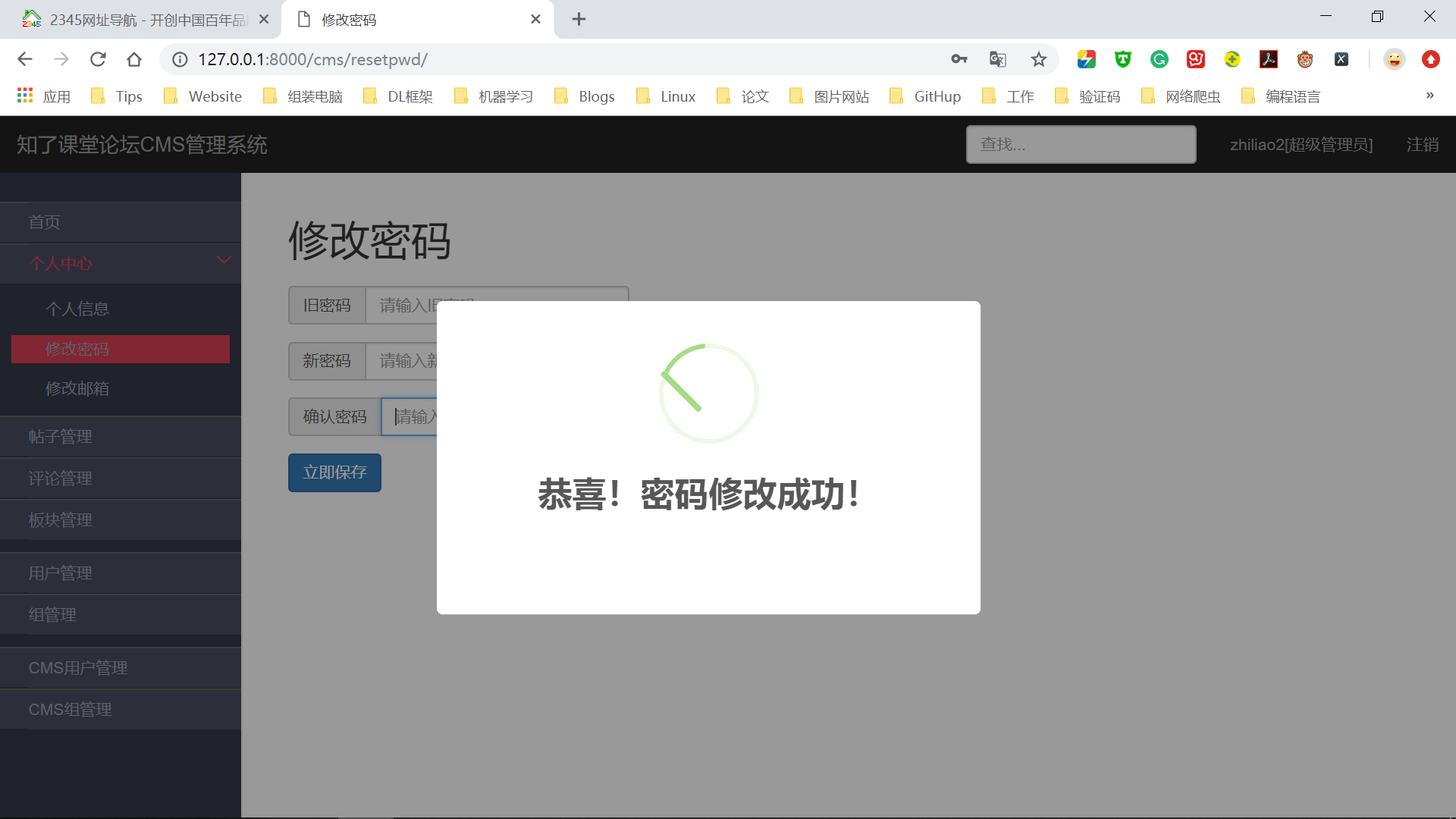
1. 在cms后台的父模板中导入这些工具包：

{# 导入zlajax.js #}  
 <script src="{{ url\_for('static', filename='common/zlajax.js') }}"></script>  
 {# 导入sweetalert工具 #}  
 <link rel="stylesheet" href="{{ static('common/sweetalert/sweetalert.css') }}">  
 <script src="{{ static('common/sweetalert/sweetalert.min.js') }}"></script>  
 <script src="{{ static('common/sweetalert/zlalert.js') }}"></script>  
 {% block head %}{% endblock %}  
</head>

1. 修改cms后台修改密码的js文件static.cms.js.resetpwd.js:

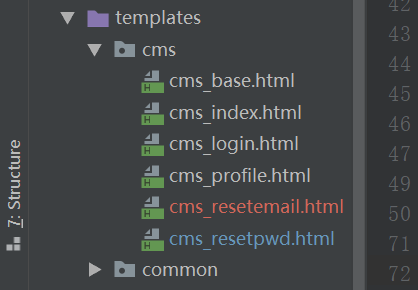
zlajax.post({  
 'url': '/cms/resetpwd/',  
 'data': {  
 'oldpwd': oldpwd,  
 'newpwd': newpwd,  
 'newpwd2': newpwd2  
 },  
 'success': **function** (data) {  
 // console.log(data);  
 **if** (data['code'] == 200) {  
 zlalert.alertSuccessToast("恭喜！密码修改成功！");  
 // 清除输入框中的内容  
 oldpwdE.val("");  
 newpwdE.val("");  
 newpwd2E.val("");  
 } **else** {  
 **var** message = data['message'];  
 zlalert.alertInfo(message)  
 }  
 },  
 'fail': **function** (error) {  
 // console.log(error);  
 zlalert.alertNetworkError();  
 }

1. 启动服务，输入新的密码，验证结果，出现以下提示框，并且输入框里的内容也被清除掉了：



# cms后台修改邮箱页面完成

1. 创建修改邮箱页面模板文件templates.cms.cms\_resetemail.html，继承父模板，并且添加BootStrap表单：



cms\_resetemail.html代码：

{% extends 'cms/cms\_base.html' %}  
  
{% block title -%}  
 修改邮箱  
{%- endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title() }}  
{% endblock %}  
  
{% block main\_content %}  
 <form action="" method="post">  
 <div class="form-container">  
 <div class="form-group">  
 <div class="input-group">  
 <input type="email" class="form-control" name="email" placeholder="请输入新邮箱">  
 <span class="input-group-addon" id="captcha-btn" style="cursor: pointer;">获取验证码</span>  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="input-group">  
 <input type="text" class="form-control" name="captcha" placeholder="请输入邮箱验证码">  
 </div>  
 </div>

<div class="form-group">  
 <button class="btn btn-primary" id="submit">立即修改</button>  
 </div>  
 </div>  
 </form>  
{% endblock %}

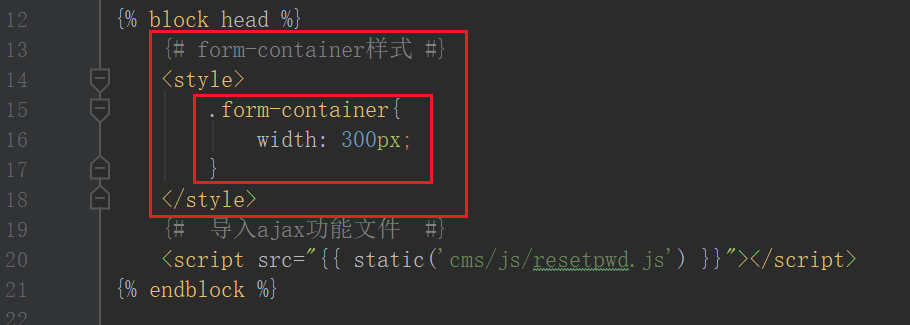
1. 在cms后台视图文件views.py中创建修改邮箱类视图，并添加到url规则中：

# 修改邮箱类视图  
class ResetEmailView(views.MethodView):  
 decorators = [login\_required] # 保证用户登陆  
  
 def get(self):  
 return render\_template('cms/cms\_resetemail.html')  
  
 def post(self):  
 pass  
  
bp.add\_url\_rule('/login/', view\_func=LoginView.as\_view('login'))  
bp.add\_url\_rule('/resetpwd/', view\_func=ResetPwdView.as\_view('resetpwd'))  
bp.add\_url\_rule('/resetemail/', view\_func=ResetEmailView.as\_view('resetemail'))

1. 添加修改邮箱页面的链接到cms父模板中：

<li class="profile-li">  
 <a href="#">个人中心<span></span></a>  
 <ul class="subnav">  
 <li><a href="{{ url\_for('cms.profile') }}">个人信息</a></li>  
 <li><a href="{{ url\_for('cms.resetpwd') }}">修改密码</a></li>  
 <li><a href="{{ url\_for('cms.resetemail') }}">修改邮箱</a></li>  
 </ul>  
</li>

1. 把cms后台修改密码模板cms\_resetpwd.html中的以下样式，剪切到静态static.cms.css.cms\_base.css父样式文件中去，因为多个地方都可以用到这个样式：



cms\_base.css添加部分代码如下：

.top-group {  
 padding: 5px 10px;  
 border-radius: 2px;  
 background: #ecedf0;  
 overflow: hidden;  
}  
  
/\* form-container样式 \*/  
.form-container {  
 width: 300px;  
}

1. 登陆打开修改邮箱页面，验证结果如下：



# Flask-Mail的使用以及邮箱配置

1. 在环境flask-env中安装Flask-Mail包和在exts.py中导入该工具:

该包的使用类似于flask\_sqlalchemy数据库， [官方文档链接](https://pythonhosted.org/Flask-Mail/)：

安装：pip install Flask-Mail

导入：from flask\_mail import Mail

exts.py添加代码，创建mail对象：

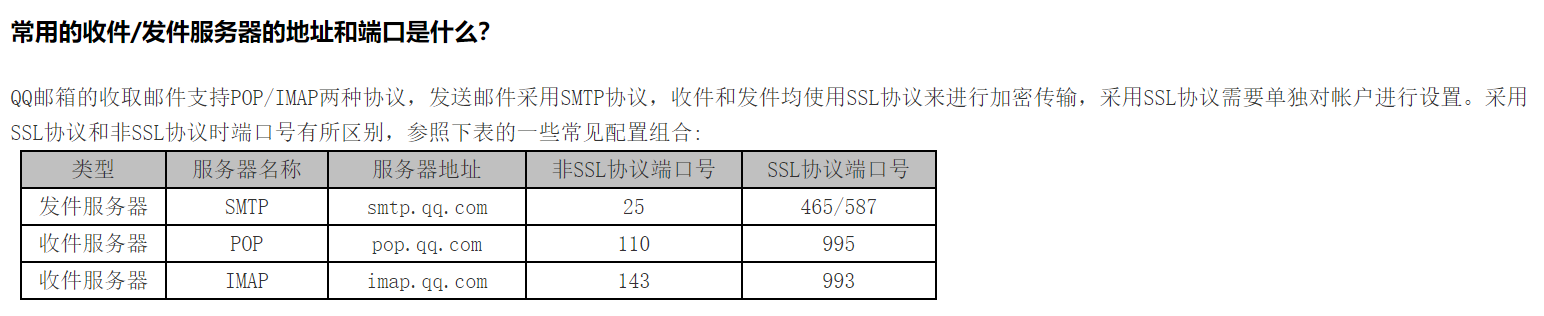
from flask\_sqlalchemy import SQLAlchemy  
from flask\_mail import Mail **导入Mail类**  
"""  
防止循环利用的exts  
"""  
  
db = SQLAlchemy()  
mail = Mail() **创建mail对象**

1. 在app.py中导入mail对象，并绑定到app中：

from exts import db, mail **导入mail**  
from flask\_wtf import CSRFProtect  
import config  
  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
 app.config.from\_object(config)  
  
 app.register\_blueprint(cms\_bp)  
 app.register\_blueprint(front\_bp)  
 app.register\_blueprint(common\_bp)  
  
 db.init\_app(app)  
 mail.init\_app(app) **mail绑定到app中** CSRFProtect(app) # 添加CSRF（跨站域请求伪造cross site request forgery）保护，防止漏洞  
  
 return app

1. 在配置文件config.py中添加发送者邮箱的服务器地址的参数：

点开链接[QQ邮箱收件/发件服务器的地址和端口](https://service.mail.qq.com/cgi-bin/help?subtype=1&id=20010&no=1000557)，查找到下图所示发送邮件的端口号：



在个人qq邮箱设置页面，获取授权码：



然后再config.py中添加以下qq邮箱服务器地址以及邮箱配置参数：

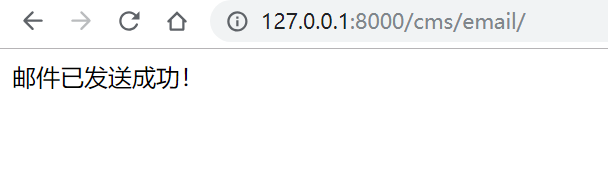
# 发送者邮箱的服务器地址和邮箱配置参数，这里使用qq邮箱  
MAIL\_SERVER = 'smtp.qq.com'  
MAIL\_PORT = '587'  
MAIL\_USE\_TLS = True  
MAIL\_USERNAME = '773985366@qq.com'  
MAIL\_PASSWORD = '。。。' **授权码**  
MAIL\_DEFAULT\_SENDER = '773985366@qq.com'

1. 再cms的视图文件views.py中，创建发送邮件视图：

from exts import db, mail  
from flask\_mail import Message

…  
# 测试发送邮件视图  
@bp.route('/email/')  
@login\_required  
def send\_email():  
 message = Message('邮件发送', recipients=['huanganfa66@163.com'], body='测试')  
 mail.send(message)  
 return '邮件已发送成功！'

1. 启动项目服务，登陆，然后打开修改邮箱页面，点击发送邮件，出现以下页面，[同时收件邮箱huanganfa66@163.com](mailto:同时收件邮箱huanganfa66@163.com)会收到一封由773985366@qq.com发出的测试邮件：

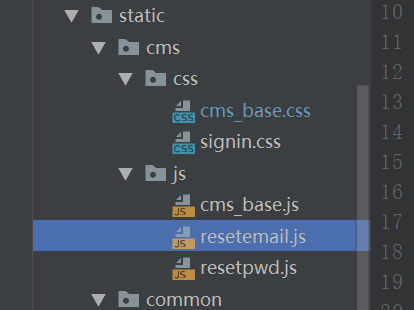


# cms后台发送邮箱验证码

1. 在cms后台视图文件views.py中创建发送邮箱验证码的视图函数：

# 发送邮件验证码视图  
@bp.route('/email\_captcha/')  
@login\_required  
def email\_captcha():  
 # /email\_captcha/?email=xxx.@qq.com 获取到发送邮箱  
 email = request.args.get('email')  
 if not email:  
 return restful.params\_error('请传递邮箱参数！')  
  
 # 随机生成6位验证码字串,大小写字母加上0-9数字  
 source = list(string.ascii\_letters)  
 source.extend(map(lambda x: str(x), range(0, 10)))  
 captcha = "".join(random.sample(source, 6))  
  
 # 给获取到的新的邮箱发送邮件验证码  
 message = Message('知了论坛邮箱验证码', recipients=[email], body='你的验证码是：%s' % captcha)  
 try:  
 mail.send(message)  
 except:  
 return restful.server\_error()  
  
 return restful.success()

1. 在静态static/cms/js下创建前端修改邮箱的js文件resetemail.js:



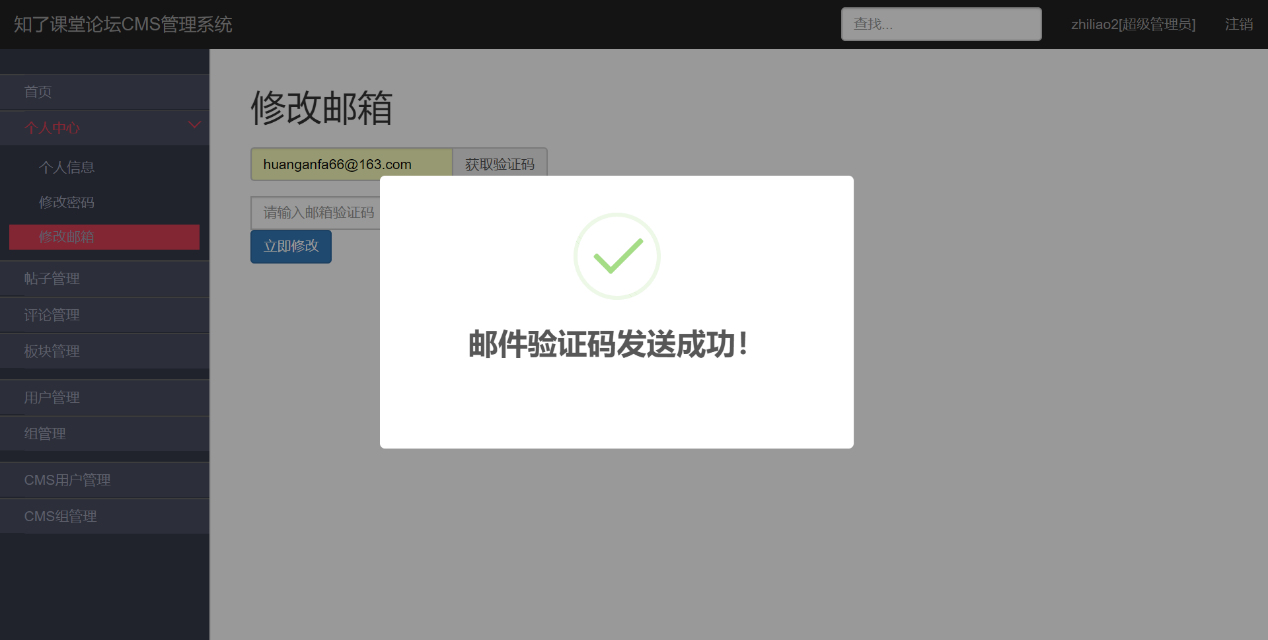
Static/cms/js/resetemail.js代码如下

// 前端修改邮箱发送验证码  
$(**function** () {  
 $('#captcha-btn').click(**function** (event) {  
 event.preventDefault();  
  
 **var** email = $("input[name='email']").val()  
 **if** (!email) {  
 zlalert.alertInfoToast('请输入新的邮箱！');  
 **return**;  
 }  
 zlajax.get({  
 'url': '/cms/email\_captcha/',  
 'data': {  
 'email': email  
 },  
 'success': **function** (data) {  
 **if** (data['code'] == 200) {  
 zlalert.alertSuccessToast('邮件验证码发送成功！');  
 } **else** {  
 zlalert.alertInfo(data['message']);  
 }  
 },  
 'fail': **function** (error) {  
 zlalert.alertNetworkError();  
 }  
 });  
 });  
});

1. 在cms后台发送邮件的模板中导入以上resetemail.js文件：

{% extends 'cms/cms\_base.html' %}  
{% from 'common/\_macros.html' **import** static %} **导入宏**  
  
{% block title -%}  
 修改邮箱  
{%- endblock %}  
  
{% block head %}  
 <script src="{{ static('cms/js/resetemail.js') }}"></script> **导入resetemail.js**  
{% endblock %}

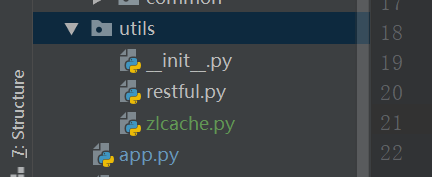
1. 启动服务，登陆，并修改邮箱，页面出现提示就说明成功：



# cms后台修改邮箱功能完成

1. 创建memcache缓存机制：

由于验证码具有时效性，所以验证验证码时可以使用memcache缓存机制。如下，在utils文件夹中创建zlcache.py:



utils/zlcache.py代码如下：

import memcache  
  
"""  
创建memcahce缓存机制  
"""  
  
cache = memcache.Client(['192.168.1.114:11211'], debug=True)  
  
def set(key, value, timeout=0):  
 return cache.set(key, value, timeout)  
  
def get(key):  
 return cache.get(key)  
  
def delete(key):  
 return cache.delete(key)

1. 在cms视图文件中添加验证码缓存：

from utils import restful, zlcache

…

# 给获取到的新的邮箱发送邮件验证码  
message = Message('知了论坛邮箱验证码', recipients=[email], body='你的验证码是：%s' % captcha)  
try:  
 mail.send(message)  
except:  
 return restful.server\_error()  
zlcache.set(email, captcha) # 添加验证码缓存  
return restful.success()

1. 在cms表单验证文件forms.py中创建修改邮箱验证表单类：

# 修改邮箱表单验证类  
class ResetEmailForm(BaseForm):  
 email = StringField(validators=[Email(message='请输入正确格式的新邮箱！')])  
 captcha = StringField(validators=[Length(min=6, max=6, message='请输入正确长度的验证码！')])  
  
 def validate\_captcha(self, field):  
 captcha = field.data # 获取用户输入的验证码  
 email = self.email.data  
 captcha\_cache = zlcache.get(email) # 获取缓存中的验证码  
 if not captcha\_cache or captcha.lower() != captcha\_cache.lower():  
 raise ValidationError('验证码有误！')  
  
 def validate\_email(self, field):  
 email = field.data  
 user = g.cms\_user  
 if user.email == email:  
 raise ValidationError('不能修改位旧的邮箱！')

1. 在cms视图文件views.py的修改邮箱类视图中添加后台验证和修改邮箱的服务代码：

from .forms import LoginForm, ResetpwdForm, ResetEmailForm

…

# 修改邮箱类视图  
class ResetEmailView(views.MethodView):  
 decorators = [login\_required] # 保证用户登陆  
  
 def get(self):  
 return render\_template('cms/cms\_resetemail.html')  
  
 def post(self):  
 form = ResetEmailForm(request.form)  
 if form.validate():  
 email = form.email.data  
 user = g.cms\_user  
 user.email = email  
 db.session.commit()  
 return restful.success()  
 else:  
 return restful.params\_error(form.get\_error())

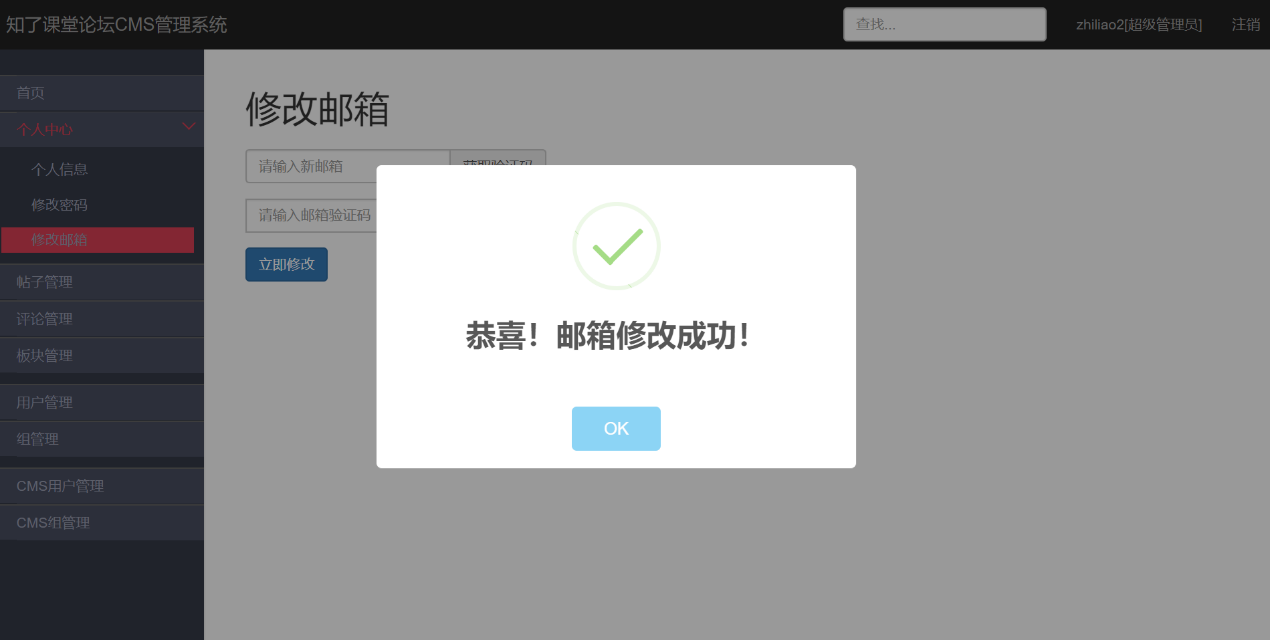
1. 在静态static/cms/js/resetemail.js中，添加前端修改邮箱验证的代码：

// 前端修改邮箱验证  
$(**function** () {  
 $('#submit').click(**function** (envent) {  
 event.preventDefault();  
  
 **var** emailE = $("input[name='email']");  
 **var** captchaE = $("input[name='captcha']");  
  
 **var** email = emailE.val();  
 **var** captcha = captchaE.val();  
  
 zlajax.post({  
 'url': '/cms/resetemail/',  
 'data': {  
 'email': email,  
 'captcha': captcha  
 },  
 'success': **function** (data) {  
 **if** (data['code'] == 200) {  
 emailE.val("");  
 captchaE.val("");  
 zlalert.alertSuccessWithTitle('恭喜！邮箱修改成功！');  
 } **else** {  
 zlalert.alertInfo(data['message']);  
 }  
 },  
 'fail': **function** (error) {  
 zlalert.alertNetworkError();  
 }  
 });  
 });  
});

1. 启动服务，登陆，并修改邮箱，出现以下提示就表示成功：

需要注意的是，如果是使用别的机器作为memcached服务端，那么必须要先开启memcached并且允许远程访问：

开启memcached服务：**/usr/bin/memcached -u thinkr -l 0.0.0.0 -p 11221 -d start**



记得分别验证输入旧邮箱、输入错误验证码的情况。

# 权限和角色模型定义

1. 如何判断某个用户有没有某个权限？

只要将需要对比的用户的权限和a权限的二进制代码进行**与**运算，如果得到的结果和a相等，就说明这个用户有a这个权限，否者就没有。

1. 在cms后台的ORM模型文件apps/cms/models.py中定义一个用户权限类，使用二进制代码代表各种不同用户的权限， 然后定义角色模型，并且定义一个关联用户和角色的中间表，代码如下：

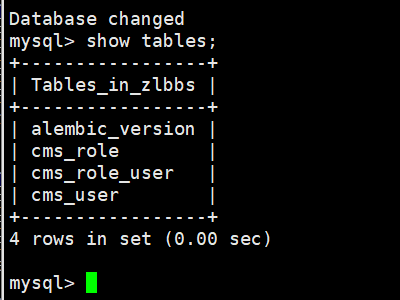
from exts import db  
from datetime import datetime  
from werkzeug.security import generate\_password\_hash, check\_password\_hash  
  
# cms权限二进制模型  
class CMSPermission(object):  
 # 0.所有权限，255的二进制表示 1111 1111  
 ALL\_PERMISSION = 0b11111111  
 # 1.访问者权限  
 VISITOR = 0b00000001  
 # 2.管理帖子权限  
 POSTER = 0b00000010  
 # 3.管理评论权限  
 COMMENTER = 0b00000100  
 # 4.管理板块权限  
 BOARDER = 0b00001000  
 # 5.管理前台用户权限  
 FRONTUSER = 0b00010000  
 # 6.管理后台用户权限  
 CMSUSER = 0b00100000  
 # 7.管理后台管理员的权限  
 ADMINER = 0b01000000  
  
# 用户和角色多对多相关联的中间表  
cms\_role\_user = db.Table(  
 'cms\_role\_user',  
 db.Column('cms\_role\_id', db.Integer, db.ForeignKey('cms\_role.id'), primary\_key=True),  
 db.Column('cms\_user\_id', db.Integer, db.ForeignKey('cms\_user.id'), primary\_key=True)  
)  
  
# cms用户模型  
class CMSUser(db.Model):  
 \_\_tablename\_\_ = 'cms\_user'  
 id = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 username = db.Column(db.String(50), nullable=False)  
 # 对外字段名叫做password  
 \_password = db.Column(db.String(100), nullable=False)  
 email = db.Column(db.String(50), nullable=False, unique=True)  
 join\_time = db.Column(db.DateTime, default=datetime.now)  
  
 # 重写初始化函数  
 def \_\_init\_\_(self, username, password, email):  
 self.username = username  
 self.password = password  
 self.email = email  
  
 # 对内的字段名叫做\_password  
 @property  
 def password(self):  
 return self.\_password  
  
 # 加密密码  
 @password.setter  
 def password(self, raw\_password):  
 self.\_password = generate\_password\_hash(raw\_password)  
  
 # 验证密码  
 def check\_password(self, raw\_password):  
 result = check\_password\_hash(self.password, raw\_password)  
 return result  
  
# 定义角色模型  
class CMSRole(db.Model):  
 \_\_tablename = 'cms\_role'  
 id = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 name = db.Column(db.String(50), nullable=False)  
 desc = db.Column(db.String(200), nullable=False)  
 create\_time = db.Column(db.DateTime, default=datetime.now)  
 permission = db.Column(db.Integer, default=CMSPermission.VISITOR)  
  
 # 关联用户和角色两个表  
 users = db.relationship('CMSUser', secondary=cms\_role\_user, backref='roles')

1. 在cmd控制台把以上模型映射到数据库：

python manage.py db migrate

python manage.py db upgrade

查看zlbbs数据可以看到创建表成功：



1. 在模型映射管理文件manage.py中，定义一个命令行创建角色的函数：

CMSUser = cms\_models.CMSUser  
CMSRole = cms\_models.CMSRole 角色模型对象  
CMSPermission = cms\_models.CMSPermission 权限模型对象

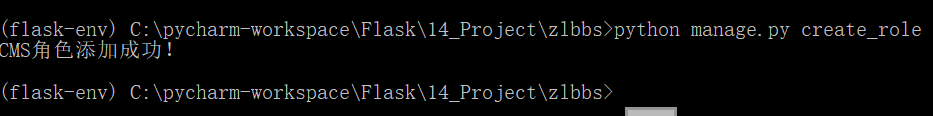
…

# 命令行创建角色函数  
@manager.command  
def create\_role():  
 # 1.访问者（可以修改个人信息）  
 visitor = CMSRole(name='访问者', desc='只能查看数据，不能修改')  
 visitor.permissions = CMSPermission.VISITOR  
  
 # 2.运营者（修改个人信息，管理帖子、管理评论、管理前台用户）  
 operator = CMSRole(name='运营者', desc='管理帖子、管理评论、管理前台用户')  
 operator.permissions = CMSPermission.VISITOR | CMSPermission.POSTER | CMSPermission.COMMENTER | CMSPermission.FRONTUSER  
  
 # 3.管理员（拥有绝大部分权限）  
 admin = CMSRole(name='管理员', desc='拥有本系统所有权限')  
 admin.permissions = CMSPermission.VISITOR | CMSPermission.POSTER | CMSPermission.COMMENTER | CMSPermission.FRONTUSER | CMSPermission.CMSUSER  
  
 # 4.开发者  
 developer = CMSRole(name='开发者', desc='开发人员专用')  
 developer.permissions = CMSPermission.ALL\_PERMISSION  
  
 db.session.add\_all([visitor, operator, admin, developer])  
 db.session.commit()

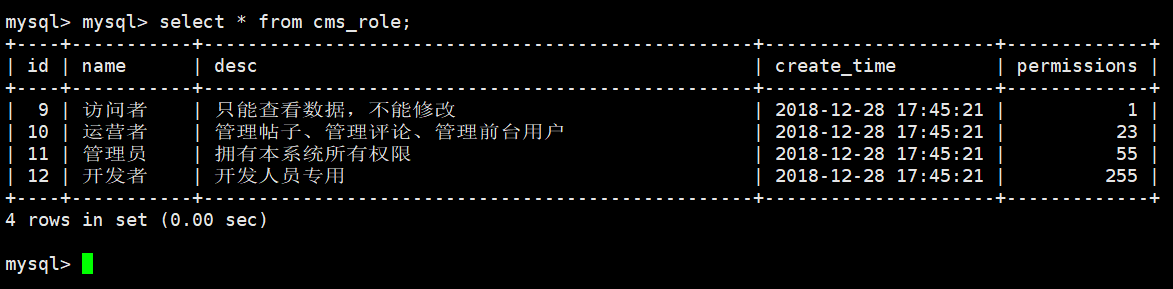
print('CMS角色添加成功！')

1. 在cmd控制台，使用命令创建角色：

python manage.py create\_role



查看表cms\_role添加结果：



# 权限功能判断完成

1. 在cms后台的模型文件models.py中往用户模型添加判断具有哪些权限的功能：

# cms用户模型  
class CMSUser(db.Model):  
 \_\_tablename\_\_ = 'cms\_user'  
 id = db.Column(db.Integer, primary\_key=True, autoincrement=True)  
 username = db.Column(db.String(50), nullable=False)  
   
…  
 # 判断具有哪些权限  
 @property  
 def permissions(self):  
 if not self.roles:  
 return 0  
 all\_permissions = 0  
 for role in self.roles:  
 permissions = role.permission  
 all\_permissions |= permissions  
 return all\_permissions  
  
 # 判断是否具有某一权限  
 def has\_permission(self, permission):  
 # all\_permission = self.permissions  
 # result = all\_permission & permission == permission # 做与运算  
 # return result  
 return self.permissions & permission == permission  
  
 # 判断是否具有开发者权限

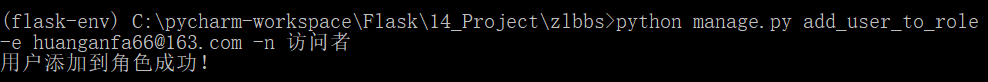
@property  
 def is\_developer(self):  
 return self.has\_permission(CMSPermission.ALL\_PERMISSION)

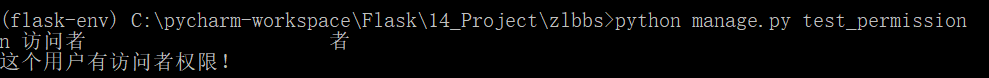
1. 在模型映射管理文件manage.py中定义命令行添加用户权限的函数以及命令行测试用户权限的函数：

# 命令行添加用户某一权限的函数  
@manager.option('-e', '--email', dest='email')  
@manager.option('-n', '--name', dest='name')  
def add\_user\_to\_role(email, name):  
 user = CMSUser.query.filter\_by(email=email).first()  
 if user:  
 role = CMSRole.query.filter\_by(name=name).first()  
 if role:  
 role.users.append(user)  
 db.session.commit()  
 print('用户添加到角色成功！')  
 else:  
 print('没有这个角色：%s' % role)  
 else:  
 print('邮箱没有这个用户：%s' % email)  
  
# 命令行测试用户是否具有某一权限的函数  
@manager.option('-n', '--name', dest='name')

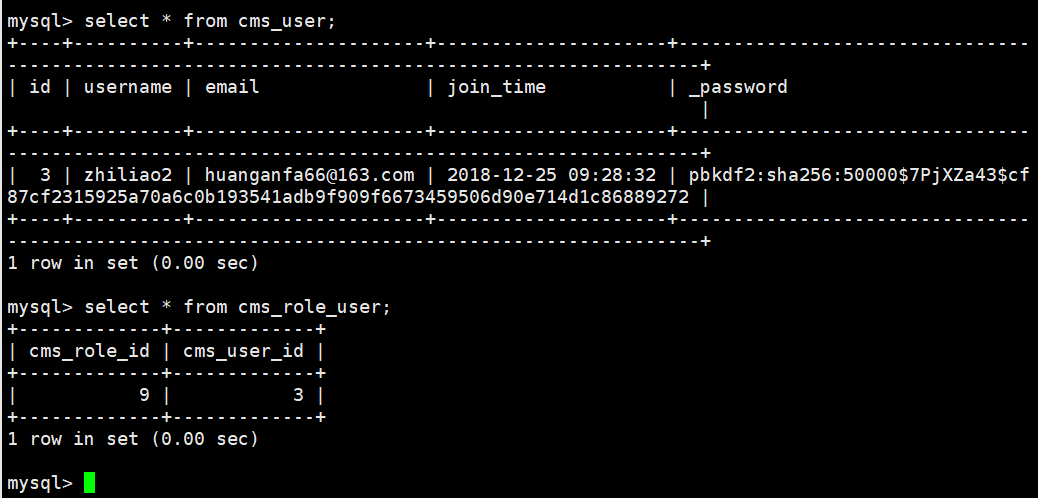
@manager.option('-e', '--email', dest='email')  
def test\_permission(name，email):  
 user = CMSUser.query.filter\_by(email=email).first()  
 if user:  
 if name == '访问者':  
 if user.has\_permission(CMSPermission.VISITOR):  
 print('这个用户有访问者权限！')  
 else:  
 print('这个用户没有访问者权限！')  
 elif name == '开发者':  
 if user.is\_developer:  
 print('这个用户有开发者权限！')  
 else:  
 print('这个用户没有开发者权限！')  
 elif name == '运营者':  
 if user.has\_permission(CMSPermission.POSTER):  
 print('这个用户有运营者权限！')  
 else:  
 print('这个用户没有运营者权限！')  
 elif name == '管理员':  
 if user.has\_permission(CMSPermission.CMSUSER):  
 print('这个用户有管理员权限！')  
 else:  
 print('这个用户没有管理员权限！')  
 else:  
 print('没有这个用户！')

1. 在cmd控制台添加权限给用户，并且测试添加结果：





查找数据库表cms\_role\_user：

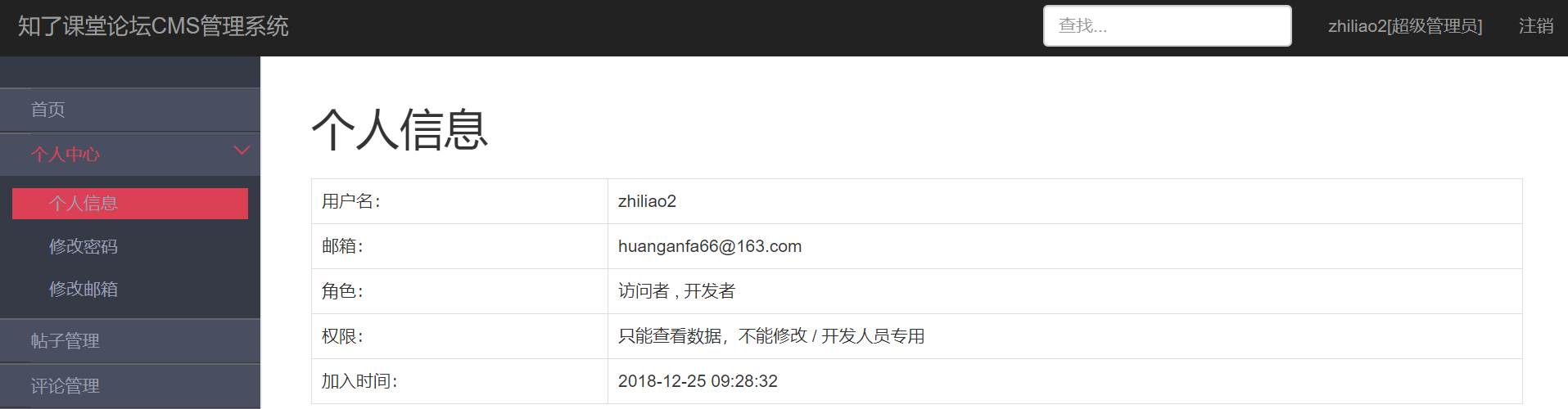


# 客户端权限验证功能完成

1. 个人详情页面角色和权限表单展示实现：

<tr>  
 <td>角色：</td>  
 <td>  
 {% for foo **in** user.roles %}  
 {{ role,name }}  
 {% if **not** loop.last %},{% endif %}  
 {% endfor %}  
 </td>  
</tr>  
<tr>  
 <td>权限：</td>  
 <td>  
 {% for foo **in** user.roles %}  
 {{ role.desc }}  
 {% if **not** loop.last %}/{% endif %}  
 {% endfor %}  
 </td>  
</tr>

启动服务，登陆，验证结果：



1. 修改用户标签展示，实现点击用户标签跳转到个人详情页面：

<div id="navbar" class="navbar-collapse collapse">  
 <ul class="nav navbar-nav navbar-right">  
 <li><a href="{{ url\_for('cms.profile') }}">{{ g.cms\_user.username }}</a></li> 修改  
 <li><a href="{{ url\_for('cms.logout') }}">注销</a></li>  
 </ul>  
 <form class="navbar-form navbar-right">  
 <input type="text" class="form-control" placeholder="查找...">  
 </form>  
</div>

1. 实现点击首页跳转到主页

<ul class="nav-sidebar">  
 <li class="unfold"><a href="{{ url\_for('cms.index') }}">首页</a></li> 修改  
 <li class="profile-li">  
 <a href="#">个人中心<span></span></a>

1. 添加用户访问者、运营者、管理者、开发者，并且给与相应的权限：

python manage.py create\_cms\_user -u 访问者 -e visitor@qq.com -p 123456

python manage.py create\_cms\_user -u 运营者 -e poster@qq.com -p 123456

python manage.py create\_cms\_user -u 管理者 -e admin@qq.com -p 123456

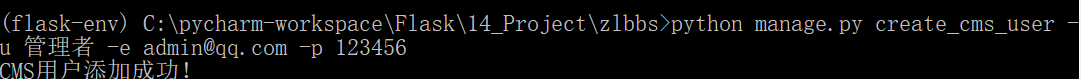
python manage.py create\_cms\_user -u 开发者 -e developer@qq.com -p 123456

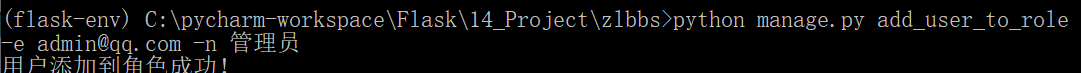
python manage.py add\_user\_to\_role -e visitor@qq.com -n 访问者

python manage.py add\_user\_to\_role -e poster@qq.com -n 运营者

python manage.py add\_user\_to\_role -e admin@qq.com -n 管理员

python manage.py add\_user\_to\_role -e developer@qq.com -n 开发者





1. 在cms/hooks.py中定义返回cms用户权限的钩子函数：

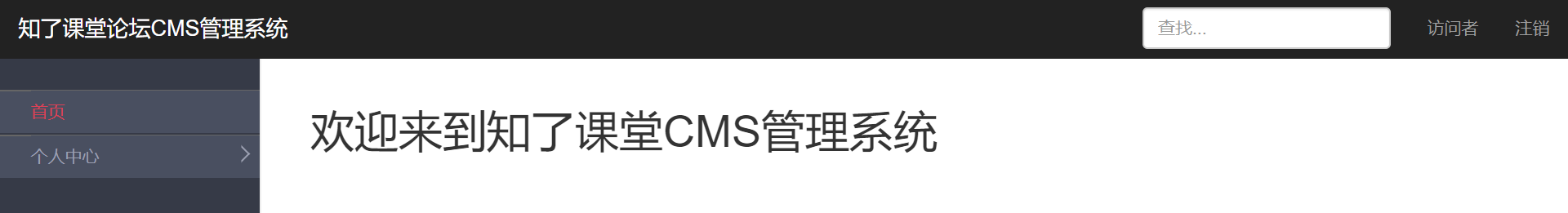
from .models import CMSUser, CMSPermission  
import config  
  
"""  
存放钩子函数  
"""  
  
# 钩子函数，线程隔离的g对象，实现cms用户名渲染  
@bp.before\_request  
def before\_request():  
 if config.CMS\_USER\_ID in session:  
 user\_id = session.get(config.CMS\_USER\_ID)  
 user = CMSUser.query.get(user\_id)  
 if user:  
 g.cms\_user = user  
   
# 返回cms用户权限的钩子函数  
@bp.context\_processor  
def cms\_context\_processor():  
 return {'CMSPermission': CMSPermission}

1. 实现前端展示用户权限列表功能：

<ul class="nav-sidebar">  
 <li class="unfold"><a href="{{ url\_for('cms.index') }}">首页</a></li>  
 <li class="profile-li">  
 <a href=" {{ url\_for('cms.profile') }}">个人中心<span></span></a> 修改  
 <ul class="subnav">  
 <li><a href="{{ url\_for('cms.profile') }}">个人信息</a></li>  
 <li><a href="{{ url\_for('cms.resetpwd') }}">修改密码</a></li>  
 <li><a href="{{ url\_for('cms.resetemail') }}">修改邮箱</a></li>  
 </ul>  
 </li>  
 {# 实现前端展示用户权限列表功能 #} 以下修改  
 {% set cms\_user = g.cms\_user %}  
 {% if cms\_user.has\_permission(CMSPermission.POSTER) %}  
 <li class="nav-group post-manage"><a href="#">帖子管理</a></li>  
 {% endif %}  
 {% if cms\_user.has\_permission(CMSPermission.COMMENTER) %}  
 <li class="comments-manage"><a href="#">评论管理</a></li>  
 {% endif %}  
 {% if cms\_user.has\_permission(CMSPermission.BOARDER) %}  
 <li class="board-manage"><a href="#">板块管理</a></li>  
 {% endif %}  
 {% if cms\_user.has\_permission(CMSPermission.FRONTUSER) %}  
 <li class="board-manage"><a href="#">前台用户管理</a></li>  
 {% endif %}  
 {% if cms\_user.has\_permission(CMSPermission.CMSUSER) %}  
 <li class="nav-group cmsuser-manage"><a href="#">CMS用户管理</a></li>  
 {% endif %}  
 {% if cms\_user.is\_developer %}  
 <li class="cmsrole-manage"><a href="#">CMS组管理</a></li>  
 {% endif %}  
</ul>

1. 启动服务，分别使用输入正确的邮箱和密码，登陆，展示各个用户的页面：

访问者



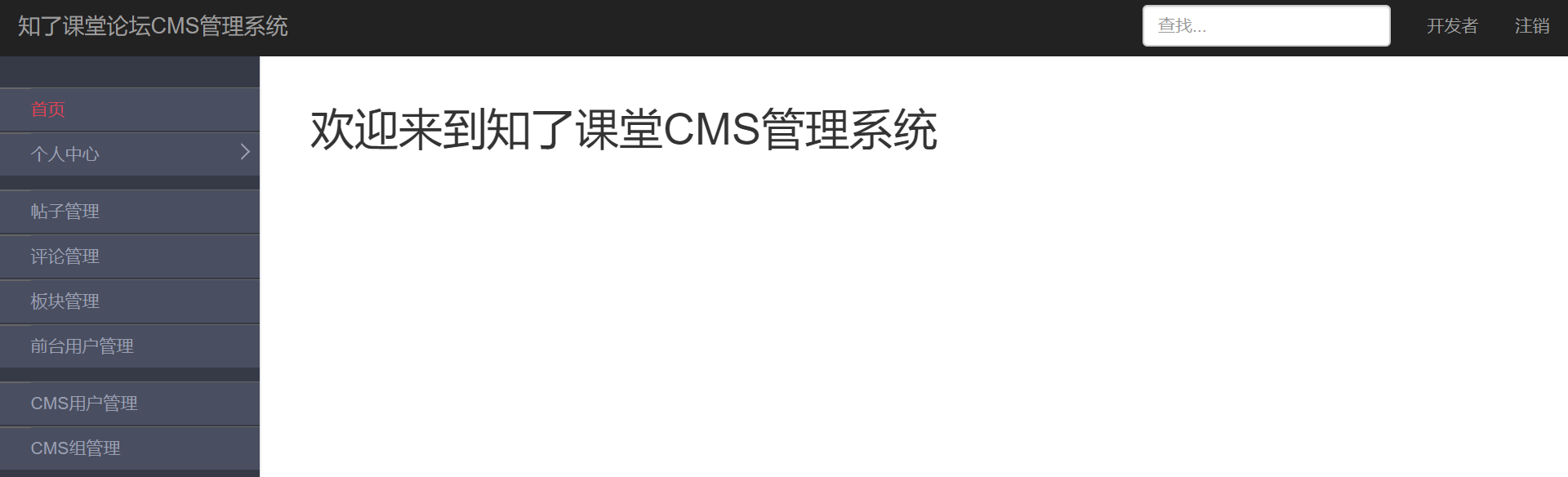
运营者



管理者



开发者



# 服务端权限验证功能完成

1. 创建cms其它几个模块的模板：

板块管理模板：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}帖子管理{% endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title }}  
{% endblock %}  
  
{% block main\_content %}  
 帖子管理页面  
{% endblock %}

评论管理模板：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}评论管理{% endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title }}  
{% endblock %}  
  
{% block main\_content %}  
 评论管理页面  
{% endblock %}

帖子管理模板：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}评论管理{% endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title }}  
{% endblock %}  
  
{% block main\_content %}  
 评论管理页面  
{% endblock %}

前台用户管理模板：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}前台用户管理{% endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title }}  
{% endblock %}  
  
{% block main\_content %}  
 前台用户管理页面  
{% endblock %}

CMS用户管理模板：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}CMS用户管理{% endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title }}  
{% endblock %}  
  
{% block main\_content %}  
 CMS用户管理页面  
{% endblock %}

CMS组管理模板：

{% extends 'cms/cms\_base.html' %}  
  
{% block title %}CMS组管理{% endblock %}  
  
{% block head %}  
  
{% endblock %}  
  
{% block page\_title %}  
 {{ self.title }}  
{% endblock %}  
  
{% block main\_content %}  
 CMS组管理页面  
{% endblock %}

1. 定义剩下各个模块的视图函数：

# 帖子管理模块视图  
@bp.route('/posts/')  
def posts():  
 return render\_template('cms/cms\_posts.html')  
  
# 评论管理模块驶入  
@bp.route('/comments/')  
def comments():  
 return render\_template('cms/cms\_comments.html')  
  
# 板块管理模块视图  
@bp.route('/boards/')  
def boards():  
 return render\_template('cms/cms\_boards.html')  
  
# 前台用户管理模块视图  
@bp.route('/fusers/')  
def fusers():  
 return render\_template('cms/cms\_fusers.html')  
  
# CMS用户管理视图  
@bp.route('/cusers/')  
def cusers():  
 return render\_template('cms/cms\_cusers.html')  
  
# CMS组管理视图  
@bp.route('/croles/')  
def croles():  
 return render\_template('cms/cms\_croles.html')

1. 修改静态static/cms/js/cms\_base.js文件，保持url一致：

$(**function** () {  
 **var** url = window.location.href;  
 **if** (url.indexOf('profile') >= 0) {  
 **var** profileLi = $('.profile-li');  
 profileLi.addClass('unfold').siblings().removeClass('unfold');  
 profileLi.children('.subnav').children().eq(0).addClass('active').siblings().removeClass('active');  
 } **else if** (url.indexOf('resetpwd') >= 0) {  
 **var** profileLi = $('.profile-li');  
 profileLi.addClass('unfold').siblings().removeClass('unfold');  
 profileLi.children('.subnav').children().eq(1).addClass('active').siblings().removeClass('active');  
 } **else if** (url.indexOf('resetemail') >= 0) {  
 **var** profileLi = $('.profile-li');  
 profileLi.addClass('unfold').siblings().removeClass('unfold');  
 profileLi.children('.subnav').children().eq(2).addClass('active').siblings().removeClass('active');  
 } **else if** (url.indexOf('posts') >= 0) {  
 **var** postManageLi = $('.post-manage');  
 postManageLi.addClass('unfold').siblings().removeClass('unfold');  
 } **else if** (url.indexOf('boards') >= 0) {  
 **var** boardManageLi = $('.board-manage');  
 boardManageLi.addClass('unfold').siblings().removeClass('unfold');  
 } **else if** (url.indexOf('permissions') >= 0) {  
 **var** permissionManageLi = $('.permission-manage');  
 permissionManageLi.addClass('unfold').siblings().removeClass('unfold');  
 } **else if** (url.indexOf('fusers') >= 0) {  
 **var** userManageLi = $('.fuser-manage');  
 userManageLi.addClass('unfold').siblings().removeClass('unfold');  
 } **else if** (url.indexOf('cuser') >= 0) {  
 **var** cmsuserManageLi = $('.cuser-manage');  
 cmsuserManageLi.addClass('unfold').siblings().removeClass('unfold');  
 } **else if** (url.indexOf('croles') >= 0) {  
 **var** cmsroleManageLi = $('.crole-manage');  
 cmsroleManageLi.addClass('unfold').siblings().removeClass('unfold');  
 } **else if** (url.indexOf('comments') >= 0) {  
 **var** commentsManageLi = $('.comment-manage');  
 commentsManageLi.addClass('unfold').siblings().removeClass('unfold');  
 }  
});

1. 启动服务，登陆查看结果，其中一个页面如下：



1. 在apps/cms/decorators.py装饰器文件中定义权限验证装饰器，保证只有该权限的用户才能访问该页面：

from flask import session, redirect, url\_for, g

…

# 权限验证装饰器，保证具有该权限才能访问该页面  
def permission\_required(permission):  
 def outter(func):  
 @wraps(func)  
 def inner(\*args, \*\*kwargs):  
 user = g.cms\_user  
 if user.has\_permission(permission):  
 return func(\*args, \*\*kwargs)  
 else:  
 return redirect(url\_for('cms.index')) # 没有权限就重定向回主页  
  
 return inner  
  
 return outter

1. 在apps/cms/views.py视图文件的各个模块视图中添加以上的装饰器，起到限制作用：

from .models import CMSUser, CMSPermission  
from .decorators import login\_required, permission\_required

…  
# 帖子管理模块视图  
@bp.route('/posts/')  
@login\_required  
@permission\_required(CMSPermission.POSTER)  
def posts():  
 return render\_template('cms/cms\_posts.html')  
  
# 评论管理模块驶入  
@bp.route('/comments/')  
@login\_required  
@permission\_required(CMSPermission.COMMENTER)  
def comments():  
 return render\_template('cms/cms\_comments.html')  
  
# 板块管理模块视图  
@bp.route('/boards/')  
@login\_required  
@permission\_required(CMSPermission.BOARDER)  
def boards():  
 return render\_template('cms/cms\_boards.html')  
  
# 前台用户管理模块视图  
@bp.route('/fusers/')  
@login\_required  
@permission\_required(CMSPermission.FRONTUSER)  
def fusers():  
 return render\_template('cms/cms\_fusers.html')  
  
# CMS用户管理视图  
@bp.route('/cusers/')  
@login\_required  
@permission\_required(CMSPermission.CMSUSER)  
def cusers():  
 return render\_template('cms/cms\_cusers.html')  
  
# CMS组管理视图  
@bp.route('/croles/')  
@login\_required  
@permission\_required(CMSPermission.ALL\_PERMISSION)  
def croles():  
 return render\_template('cms/cms\_croles.html')

1. 启动服务，在浏览器测试链接，如果没有权限回跳转回主页如下:



# front前台用户模型创建完成

1. 这里需要用到一个工具来生成前台用户的id，在flask-env环境中安装python包shortuuid:

pip install shortuuid

1. 在apps/front/models.py中创建前台用户的ORM模型类：

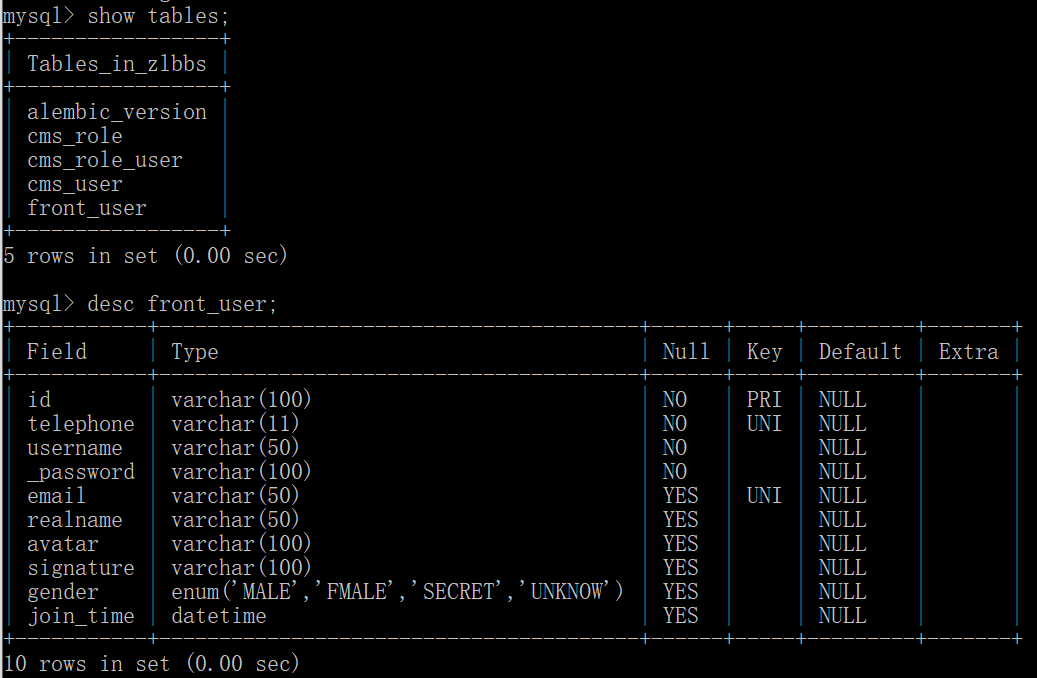
from exts import db  
from werkzeug.security import generate\_password\_hash, check\_password\_hash  
from datetime import datetime  
import shortuuid  
import enum  
  
# 性别  
class GenderEnum(enum.Enum):  
 MALE = 1  
 FMALE = 2  
 SECRET = 3  
 UNKNOW = 4  
  
# 前台用户模型  
class FrontUser(db.Model):  
 \_\_tablename\_\_ = 'front\_user'  
 id = db.Column(db.String(100), primary\_key=True, default=shortuuid.uuid) # 使用shortuuid.uuid生成id  
 telephone = db.Column(db.String(11), nullable=False, unique=True)  
 username = db.Column(db.String(50), nullable=False)  
 \_password = db.Column(db.String(100), nullable=False) # 密码比较特殊，需要加密  
 email = db.Column(db.String(50), unique=True)  
 realname = db.Column(db.String(50))  
 avatar = db.Column(db.String(100))  
 signature = db.Column(db.String(100))  
 gender = db.Column(db.Enum(GenderEnum), default=GenderEnum.UNKNOW)  
 join\_time = db.Column(db.DateTime, default=datetime.now)  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 if 'password' in kwargs:  
 self.password = kwargs.get('password')  
 kwargs.pop('password')  
 super(FrontUser, self).\_\_init\_\_(\*args, \*\*kwargs)  
  
 @property  
 def password(self):  
 return self.\_password  
  
 @password.setter  
 def password(self, newpwd):  
 self.\_password = generate\_password\_hash(newpwd)  
  
 def check\_password(self, rawpwd):  
 return check\_password\_hash(self.\_password, rawpwd)

1. 在cmd将以上模型映射文件到数据库：

>python manage.py db migrate

>python manage.py db upgrade

查看数据库zlbbs，表格front\_user创建成功：



1. 在mange.py定义命令行创建Front前台用户的映射函数：

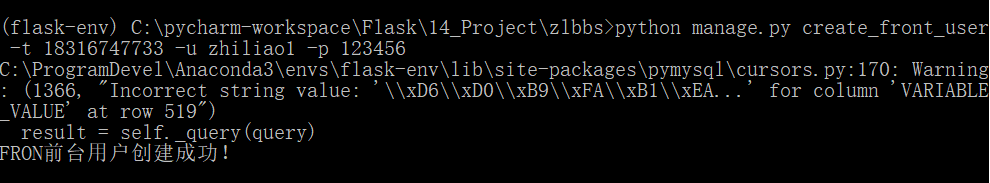
from apps.front import models as front\_models  
  
# 后台模型对象  
CMSUser = cms\_models.CMSUser  
CMSRole = cms\_models.CMSRole  
CMSPermission = cms\_models.CMSPermission  
  
# 前台模型对象  
FrontUser = front\_models.FrontUser

…

# 命令行创建前台用户映射函数  
@manager.option('-t', '--telephone', dest='telephone')  
@manager.option('-u', '--username', dest='username')  
@manager.option('-p', '--password', dest='password')  
def create\_front\_user(telephone, username, password):  
 user = FrontUser(telephone=telephone, username=username, password=password)  
 db.session.add(user)  
 db.session.commit()  
 print('FRON前台用户创建成功！')

1. 在cmd控制台，使用命令行创建前台用户：

>python manage.py create\_front\_user -t 18316747733 -u zhiliao1 -p 123456

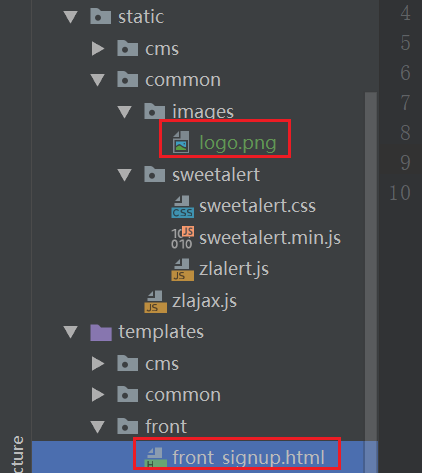


查看数据库，前台用户创建成功：



# front前台注册页面完成

1. 把logo图标文件logo.png拷贝到静态static/common/images文件夹中:

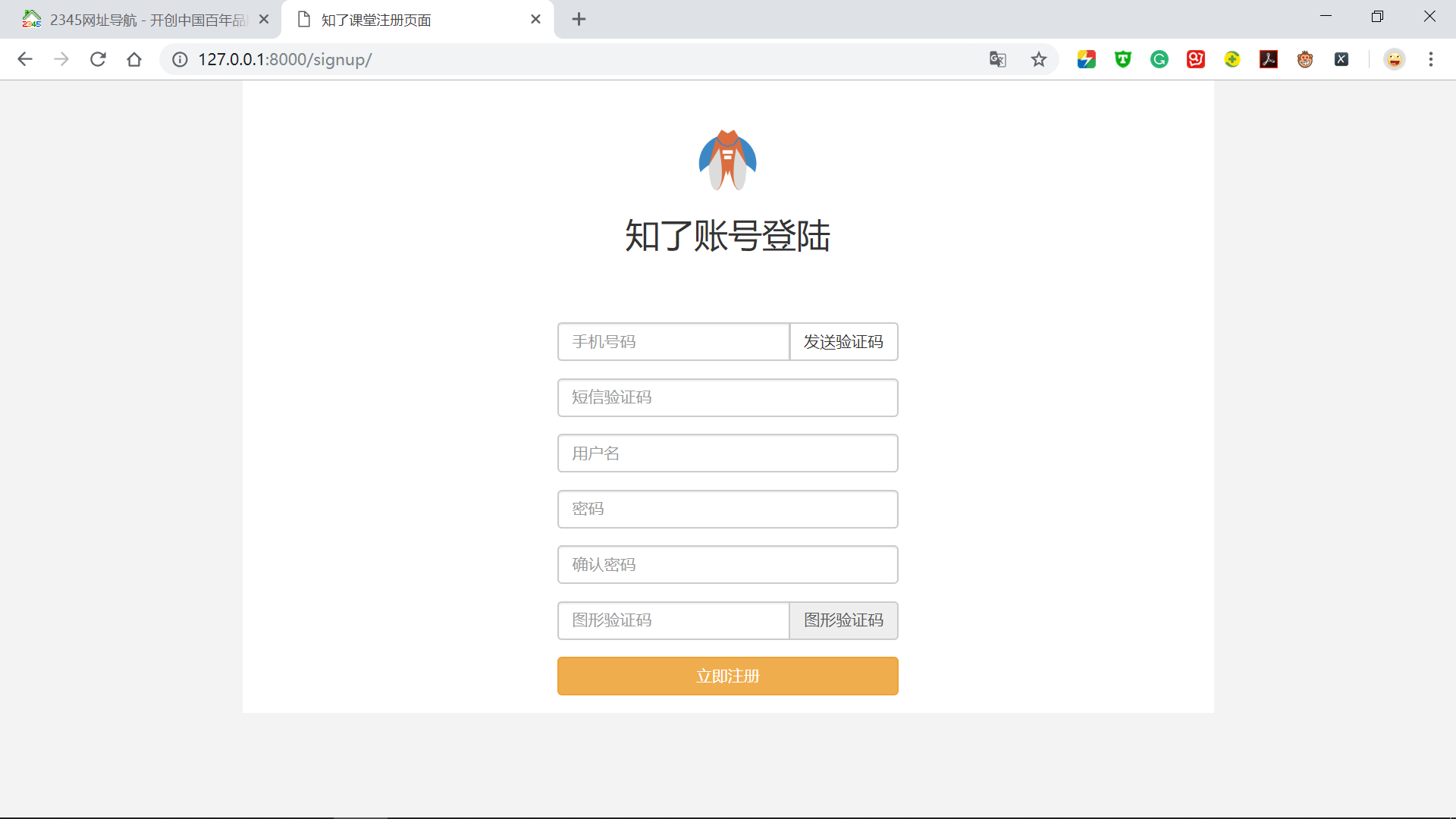


1. 创建前台注册页面模板，代码如下：

{% from 'common/\_macros.html' **import** static %}  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>知了课堂注册页面</title>  
 <script src="http://cdn.bootcss.com/jquery/3.1.1/jquery.min.js"></script>  
 <link href="http://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script src="http://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
 <style>  
 body {  
 background: #f3f3f3;  
 }  
 .outer-box {  
 width: 854px;  
 background: #fff;  
 margin: 0 auto;  
 overflow: hidden;  
 }  
 .logo-box {  
 text-align: center;  
 padding-top: 40px;  
 }  
 .logo-box img {  
 width: 60px;  
 height: 60px;  
 }  
 .page-title {  
 text-align: center;  
 }  
 .sign-box {  
 width: 300px;  
 margin: 0 auto;  
 padding-top: 50px;  
 }  
 </style>  
</head>  
<body>  
<div class="outer-box">  
 <div class="logo-box">  
 <a href="/">  
 {# 加载logo图标 #}  
 <img src="{{ static('common/images/logo.png') }}" alt="">  
 </a>  
 </div>  
 <h2 class="page-title">知了账号登陆</h2>  
 <div class="sign-box">  
 <div class="form-group">  
 <div class="input-group">  
 <input type="text" class="form-control" name="telephone" placeholder="手机号码">  
 <span class="input-group-btn">  
 <button class="btn btn-default">发送验证码</button>  
 </span>  
 </div>  
 </div>  
 <div class="form-group">  
 <input type="text" class="form-control" name="sms\_captcha" placeholder="短信验证码">  
 </div>  
 <div class="form-group">  
 <input type="text" class="form-control" name="username" placeholder="用户名">  
 </div>  
 <div class="form-group">  
 <input type="password" class="form-control" name="password1" placeholder="密码">  
 </div>  
 <div class="form-group">  
 <input type="password" class="form-control" name="password2" placeholder="确认密码">  
 </div>  
 <div class="form-group">  
 <div class="input-group">  
 <input type="text" class="form-control" name="graph\_captcha" placeholder="图形验证码">  
 <span class="input-group-addon"> 图形验证码</span>  
 </div>

</div>  
 <div class="form-group">  
 <button class="btn btn-warning btn-block">立即注册</button>  
 </div>  
 </div>  
</div>  
</body>  
</html>

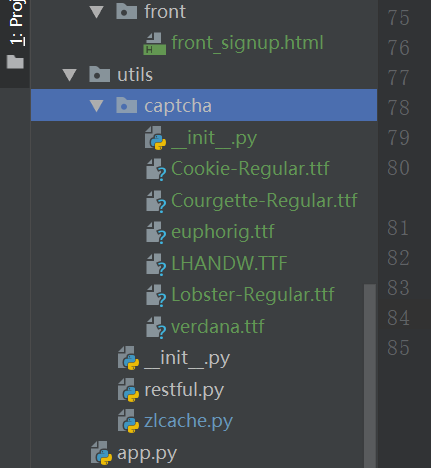
1. 启动服务，浏览器中打开链接<http://127.0.0.1:8000/signup/>，看到以下页面：



# front前台生成图形验证码

根据图形验证码生成技术已经做好了一个工具包，放入文件夹captcha中，使用的时候直接拷贝一份就好。

1. 把图形验证码生成工具包拷贝到utils文件中：



1. 首先要在环境flask-env中安装pillow：

>pip install pillow 或者 >conda install pillow

1. 图形验证码生成工具包介绍：

图形验证码生成工具包中，除了\_\_init\_\_.py文件外，其它的文件都是从windows系统中C:\Windows\Fonts拷贝来的字体文件，captcha/\_\_init\_\_.py的代码如下：

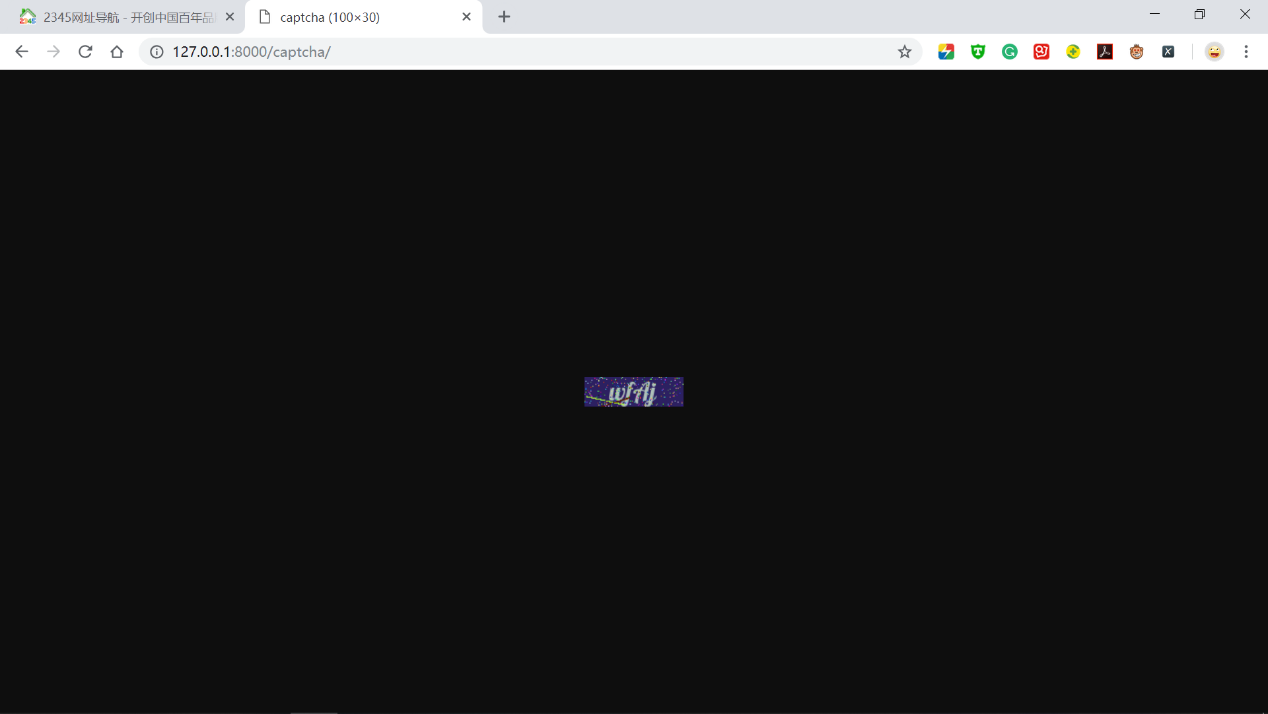
import random  
import string  
# Image：一个画布  
# ImageDraw：一个画笔  
# ImageFont:画笔的字体  
from PIL import Image, ImageDraw, ImageFont # pip install pillow  
  
# Captcha验证码  
class Captcha(object):  
 # 生成几位数的验证码  
 number = 4  
 # 验证码图片的宽度和高度  
 size = (100, 30)  
 # 验证码字体大小  
 fontsize = 25  
 # 加入干扰线的条数  
 line\_number = 2  
  
 # 构建一个验证码源文本  
 SOURCE = list(string.ascii\_letters)  
 for index in range(0, 10):  
 SOURCE.append(str(index))  
  
 # 用来绘制干扰线  
 @classmethod  
 def \_\_gene\_line(cls, draw, width, height):  
 begin = (random.randint(0, width), random.randint(0, height))  
 end = (random.randint(0, width), random.randint(0, height))  
 draw.line([begin, end], fill=cls.\_\_gene\_random\_color(), width=2)  
  
 # 用来绘制干扰点  
 @classmethod  
 def \_\_gene\_points(cls, draw, point\_chance, width, height):  
 chance = min(100, max(0, int(point\_chance))) # 大小限制在[0, 100]  
 for w in range(width):  
 for h in range(height):  
 tmp = random.randint(0, 100)  
 if tmp > 100 - chance:  
 draw.point((w, h), fill=cls.\_\_gene\_random\_color())  
  
 # 生成随机的颜色  
 @classmethod  
 def \_\_gene\_random\_color(cls, start=0, end=255):  
 random.seed()  
 return (random.randint(start, end), random.randint(start, end), random.randint(start, end))  
  
 # 随机选择一个字体  
 @classmethod  
 def \_\_gene\_random\_font(cls):  
 # 各种字体

fonts = [  
 'Courgette-Regular.ttf',  
 'LHANDW.TTF',  
 'Lobster-Regular.ttf',  
 'verdana.ttf'  
 ]  
 font = random.choice(fonts)  
 return 'utils/captcha/' + font  
  
 # 用来随机生成一个字符串(包括英文和数字)  
 @classmethod  
 def gene\_text(cls, number):  
 # number是生成验证码的位数  
 return ''.join(random.sample(cls.SOURCE, number))  
  
 # 生成验证码  
 @classmethod  
 def gene\_graph\_captcha(cls):  
 # 验证码图片的宽和高  
 width, height = cls.size  
 # 创建图片  
 # R：Red（红色）0-255  
 # G：G（绿色）0-255  
 # B：B（蓝色）0-255  
 # A：Alpha（透明度）  
 image = Image.new('RGBA', (width, height), cls.\_\_gene\_random\_color(0, 100))  
 # 验证码的字体  
 font = ImageFont.truetype(cls.\_\_gene\_random\_font(), cls.fontsize)  
 # 创建画笔  
 draw = ImageDraw.Draw(image)  
 # 生成字符串  
 text = cls.gene\_text(cls.number)  
 # 获取字体的尺寸  
 font\_width, font\_height = font.getsize(text)  
 # 填充字符串  
 draw.text(((width - font\_width) / 2, (height - font\_height) / 2), text, font=font,  
 fill=cls.\_\_gene\_random\_color(150, 255))  
 # 绘制干扰线  
 for x in range(0, cls.line\_number):  
 cls.\_\_gene\_line(draw, width, height)  
 # 绘制噪点  
 cls.\_\_gene\_points(draw, 10, width, height)  
 return (text, image)

1. 在front前台的视图文件front/views.py中定义图形验证码生成视图函数：

from flask import (  
 Blueprint,  
 views,  
 render\_template,  
 make\_response  
)  
from utils.captcha import Captcha  
from io import BytesIO  
  
bp = Blueprint('front', \_\_name\_\_)  
  
@bp.route('/')  
def index():  
 return 'front index'  
  
# 图形验证码生成视图函数  
@bp.route('/captcha/')  
def graph\_captcha():  
 text, image = Captcha.gene\_graph\_captcha()  
 out = BytesIO()  
 image.save(out, 'png')  
 out.seek(0)  
 resp = make\_response(out.read())  
 resp.content\_type = 'image/png'  
 return resp  
  
class SignupView(views.MethodView):  
 def get(self):  
 return render\_template('front/front\_signup.html')  
  
bp.add\_url\_rule('/signup/', view\_func=SignupView.as\_view('signup'))

1. 启动服务，浏览器打开链接<http://127.0.0.1:8000/captcha/>，出现图形验证码。。。



# front前台注册页面点击更新图形验证码

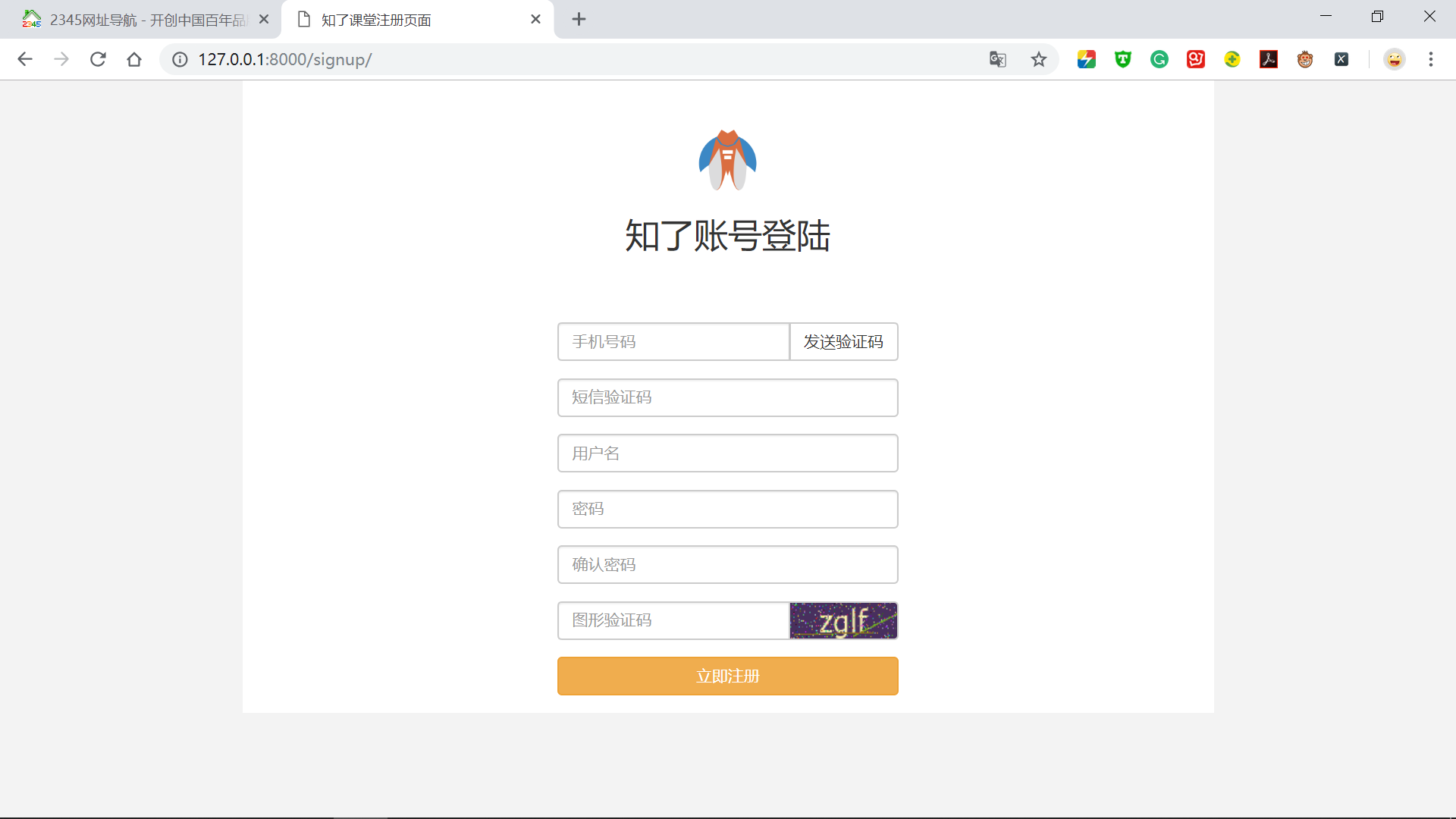
1. 在前台模板front\_signup.html中添加图形验证码生成链接：

<div class="form-group">  
 <div class="input-group">  
 <input type="text" class="form-control" name="graph\_captcha" placeholder="图形验证码">  
 <span class="input-group-addon captcha-addon">  
 <img id="captcha-img" class="captcha-img" src="{{ url\_for('front.graph\_captcha') }}" alt="">  
 </span>  
 </div>  
</div>

给验证码图片添加样式添加样式：

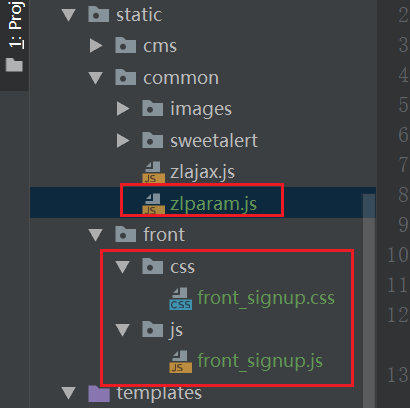
.captcha-addon{  
 padding: 0;  
 overflow: hidden;  
}  
.captcha-img{  
 width: 94px;  
 height: 32px;  
 cursor: pointer;  
}

启动服务验证，打开链接http://127.0.0.1:8000/signup/：



1. 实现验证码图片点击更新功能

把一个写好的js工具文件拷贝到static/common/zlparam.js:



zlparam.js全部代码如下：

**var** zlparam = {  
 setParam: **function** (href, key, value) {  
 // 重新加载整个页面  
 **var** isReplaced = **false**;  
 **var** urlArray = href.split('?');  
 **if** (urlArray.length > 1) {  
 **var** queryArray = urlArray[1].split('&');  
 **for** (**var** i = 0; i < queryArray.length; i++) {  
 **var** paramsArray = queryArray[i].split('=');  
 **if** (paramsArray[0] == key) {  
 paramsArray[1] = value;  
 queryArray[i] = paramsArray.join('=');  
 isReplaced = **true**;  
 **break**;  
 }  
 }  
  
 **if** (!isReplaced) {  
 **var** params = {};  
 params[key] = value;  
 **if** (urlArray.length > 1) {  
 href = href + '&' + $.param(params);  
 } **else** {  
 href = href + '?' + $.param(params);  
 }  
 } **else** {  
 **var** params = queryArray.join('&');  
 urlArray[1] = params;  
 href = urlArray.join('?');  
 }  
 } **else** {  
 **var** param = {};  
 param[key] = value;  
 **if** (urlArray.length > 1) {  
 href = href + '&' + $.param(param);  
 } **else** {  
 href = href + '?' + $.param(param);  
 }  
 }  
 **return** href;  
 }  
};

在静态文件夹static/front/js中创建前台注册js文件front\_signup.js，代码如下：

$(**function** () {  
 $('#captcha-img').click(**function** (event) {  
 **var** self = $(**this**)  
 **var** src = self.attr('src');  
 **var** newsrc = zlparam.setParam(src, 'xx', Math.random());  
  
 self.attr('src', newsrc);  
 });  
});

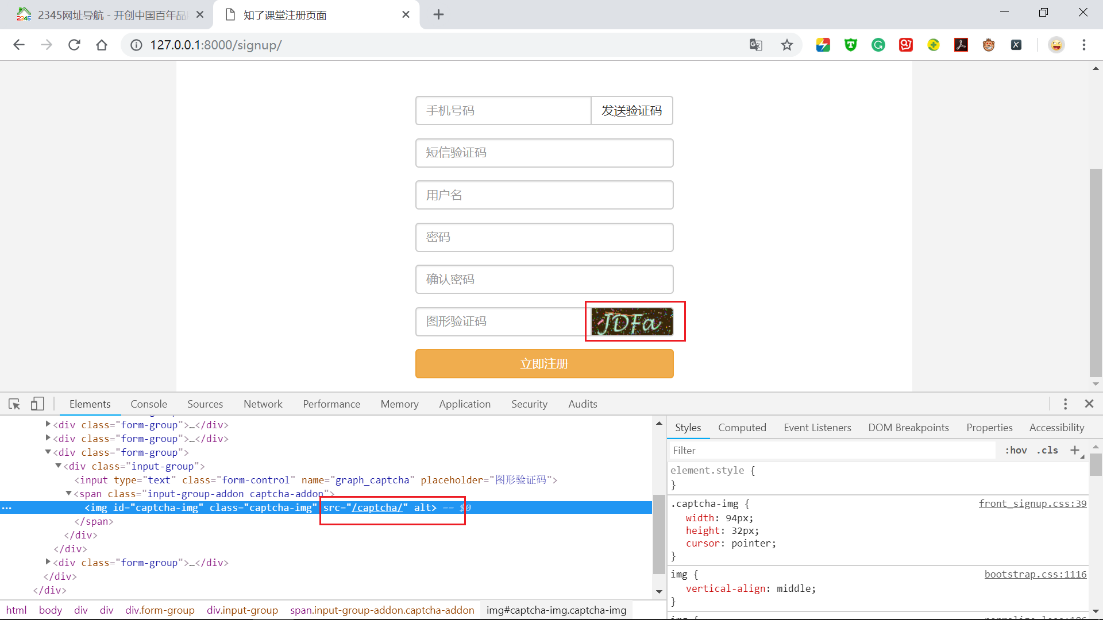
创建前台注册样式文件static/front/css/front\_signup.css，代码如下：

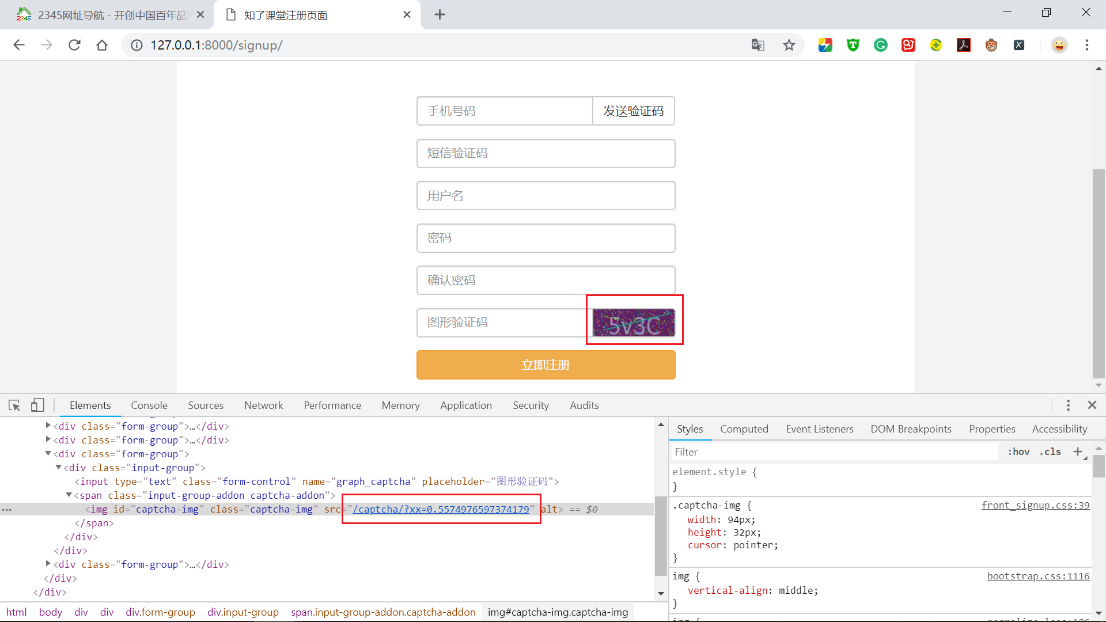
body {  
 background: #f3f3f3;  
}  
  
.outer-box {  
 width: 854px;  
 background: #fff;  
 margin: 0 auto;  
 overflow: hidden;  
}  
  
.logo-box {  
 text-align: center;  
 padding-top: 40px;  
}  
  
.logo-box img {  
 width: 60px;  
 height: 60px;  
}  
  
.page-title {  
 text-align: center;  
}  
  
.sign-box {  
 width: 300px;  
 margin: 0 auto;  
 padding-top: 50px;  
}  
  
.captcha-addon {  
 padding: 0;  
 overflow: hidden;  
}

1. 在前台注册模板front\_signup.html中加载以上三个文件：

<head>  
 <meta charset="UTF-8">  
 <title>知了课堂注册页面</title>  
 <script src="http://cdn.bootcss.com/jquery/3.1.1/jquery.min.js"></script>  
 <link href="http://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">  
 <script src="http://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  
 <script src="{{ static('common/zlparam.js') }}"></script>  
 <script src="{{ static('front/js/front\_signup.js') }}"></script>  
 <link rel="stylesheet" href="{{ static('front/css/front\_signup.css') }}">  
</head>

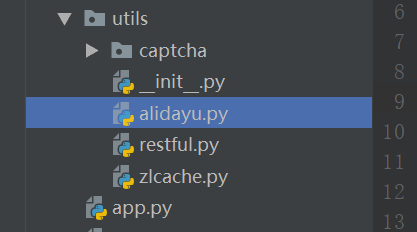
1. 启动服务，打开链接http://127.0.0.1:8000/signup/，并且点击验证码图片，验证更新功能。。。





# front前台注册页面对接短信验证码接口

1. 把阿里云大于短信验证码sdk文件alidayu.py拷贝到utils文件夹中



# 知了课堂-阿里大于短信验证码sdk  
import hashlib  
from time import time  
import logging  
import requests # pip install requests  
  
class AlidayuAPI(object):  
 APP\_KEY\_FIELD = 'ALIDAYU\_APP\_KEY'  
 APP\_SECRET\_FIELD = 'ALIDAYU\_APP\_SECRET'  
 SMS\_SIGN\_NAME\_FIELD = 'ALIDAYU\_SIGN\_NAME'  
 SMS\_TEMPLATE\_CODE\_FIELD = 'ALIDAYU\_TEMPLATE\_CODE'  
  
 def \_\_init\_\_(self, app=None):  
 self.url = 'https://eco.taobao.com/router/rest'  
 self.headers = {  
 'Content-type': 'application/x-www-form-urlencoded;charset=UTF-8',  
 "Cache-Control": "no-cache",  
 "Connection": "Keep-Alive",  
 }  
 if app:  
 self.init\_app(app)  
  
 def init\_app(self, app):  
 config = app.config  
 try:  
 self.key = config[self.APP\_KEY\_FIELD]  
 self.secret = config[self.APP\_SECRET\_FIELD]  
 self.sign\_name = config[self.SMS\_SIGN\_NAME\_FIELD]  
 self.api\_params = {  
 'sms\_free\_sign\_name': config[self.SMS\_SIGN\_NAME\_FIELD],  
 'sms\_template\_code': config[self.SMS\_TEMPLATE\_CODE\_FIELD],  
 'extend': '',  
 'sms\_type': "normal",  
 "method": "alibaba.aliqin.fc.sms.num.send",  
 "app\_key": self.key,  
 "format": "json",  
 "v": "2.0",  
 "partner\_id": "",  
 "sign\_method": "md5",  
 }  
 except Exception as e:  
 logging.error(e.args)  
 raise ValueError('请填写正确的阿里大鱼配置！')  
  
 def send\_sms(self, telephone, \*\*params):  
 self.api\_params['timestamp'] = str(int(time() \* 1000))  
 self.api\_params['sms\_param'] = str(params)  
 self.api\_params['rec\_num'] = telephone  
  
 newparams = "".join(["%s%s" % (k, v) for k, v in sorted(self.api\_params.items())])  
 newparams = self.secret + newparams + self.secret  
 sign = hashlib.md5(newparams.encode("utf-8")).hexdigest().upper()  
 self.api\_params['sign'] = sign  
  
 resp = requests.post(self.url, params=self.api\_params, headers=self.headers)  
 data = resp.json()  
 try:  
 result = data['alibaba\_aliqin\_fc\_sms\_num\_send\_response']['result']['success']  
 return result  
 except:  
 print('=' \* 10)  
 print("阿里大于错误信息：", data)  
 print('=' \* 10)  
 return False

要安装requests包：

>pip install requests

1. 在exts.py文件中创建一个阿里大于API对象

from flask\_sqlalchemy import SQLAlchemy  
from flask\_mail import Mail  
from utils.alidayu import AlidayuAPI  
  
"""  
防止循环利用的exts  
"""  
  
db = SQLAlchemy() # sqlalchemy数据库对象  
mail = Mail() # mail对象  
alidayu = AlidayuAPI() # 阿里大于api对象

1. 在app.py文件中初始化阿里大于api对象

from exts import db, mail, alidayu  
from flask\_wtf import CSRFProtect  
import config  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
 app.config.from\_object(config)  
  
 app.register\_blueprint(cms\_bp)  
 app.register\_blueprint(front\_bp)  
 app.register\_blueprint(common\_bp)  
  
 db.init\_app(app)  
 mail.init\_app(app)  
 alidayu.init\_app(app) # 初始化阿里大于api对象

1. 在config.py文件中配置阿里大于短信发送的配置参数

# 阿里大于短信验证码相关配置参数  
ALIDAYU\_APP\_KEY = '23709557'  
ALIDAYU\_APP\_SECRET = 'd9e430e0a96e21c92adacb522a905c4b'  
ALIDAYU\_SIGN\_NAME = '小饭桌应用'  
ALIDAYU\_TEMPLATE\_CODE = 'SMS\_68465012'

1. 在公共视图文件apps/common/views.py中定义短信验证码发送视图函数

from flask import Blueprint, request  
from exts import alidayu  
from utils import restful  
from utils.captcha import Captcha  
  
bp = Blueprint('common', \_\_name\_\_, url\_prefix='/common')  
# 短信验证码发送视图函数  
@bp.route('/sms\_captcha/')  
def sms\_captcha():  
 # ?telephone=xxx ===> /common/sms\_capcha/xxx  
 telephone = request.args.get('telephone')  
 if not telephone:  
 return restful.params\_error(message='请传入手机号码！')  
 captcha = Captcha.gene\_text(number=6) # 生成6位的验证码  
 if alidayu.send\_sms(telephone, code=captcha):  
 return restful.success()  
 else:  
 # return restful.params\_error(message='短信验证码发送失败！')  
 return restful.success() # 为了方便开发，假设短信发送成功

1. 在前台注册模板文件front\_signup.html中，给发送验证码按钮添加一个id，并把需要用到的文件加载进来

<head>  
 …  
 <script src="{{ url\_for('static', filename='common/zlajax.js') }}"></script>  
 <link rel="stylesheet" href="{{ static('common/sweetalert/sweetalert.css') }}">  
 <script src="{{ static('common/sweetalert/sweetalert.min.js') }}"></script>  
 <script src="{{ static('common/sweetalert/zlalert.js') }}"></script>  
 …  
</head>

<body>  
<div class="outer-box">  
 …  
 <div class="form-group">  
 <div class="input-group">  
 <input type="text" class="form-control" name="telephone" placeholder="手机号码">  
 <span class="input-group-btn">  
 <button **id="sms-captcha-btn"** class="btn btn-default">发送验证码</button>  
 </span>  
 </div>  
 </div>  
 …  
</div>  
</body>

1. 在前台的前端文件static/front/js/front\_signup.js中实现短信发送验证码功能

// 短信验证码点击发送  
$(**function** () {  
 $('#sms-captcha-btn').click(**function** (event) {  
 event.preventDefault();  
 **var** self = $(**this**);  
 **var** telephone = $("input[name='telephone']").val();  
  
 **if** (!(/^1[345879]\d{9}$/.test(telephone))) {  
 zlalert.alertInfoToast('请输入正确的手机号码！');  
 **return**;  
 }  
 zlajax.get({  
 'url': '/common/sms\_captcha?telephone=' + telephone,  
 'success': **function** (data) {  
 **if** (data['code'] == 200) {  
 zlalert.alertSuccessToast('短信验证码发送成功！');  
 self.attr('disable', 'disabled');  
 **var** timeCount = 60; // 验证码有效期60秒  
 setInterval(**function** () {  
 timeCount--;  
 self.text(timeCount);  
 }, 1000);  
 } **else** {  
 zlalert.alertInfoToast(data['message']);  
 }  
 }  
 });  
 });  
});

# front前台短信验证码接口加密&js代码压缩加密和混淆

1. 在apps/forms.py的父类中添加一个验证的函数

from wtforms import Form  
  
# 基础的表单验证类  
class BaseForm(Form):  
 def get\_error(self):  
 message = self.errors.popitem()[1][0] # 需要返回什么信息视具体情况而定  
 return message  
  
 # 验证  
 def validate(self):  
 return super(BaseForm, self).validate()

1. 在公共apps/common/forms.py中定义短信验证码验证表单类

from wtforms import StringField  
from wtforms.validators import regexp, InputRequired  
from apps.forms import BaseForm # 导入和继承基础表单验证类  
import hashlib  
  
"""  
前台表单验证  
"""  
  
# 短信验证码验证表单类  
class SMSCaptchaForm(BaseForm):  
 salt = 'hihasiyfbfwe&($^\*&hunggfaM' # 加盐  
 telephone = StringField(validators=[regexp(r'1[345789]\d{9}')])  
 timestamp = StringField(validators=[regexp(r'\d{13}')])  
 sign = StringField(validators=[InputRequired()])  
  
 def validate(self):  
 result = super(SMSCaptchaForm, self).validate()  
 if not result:  
 return False  
  
 telephone = self.telephone.data  
 timestamp = self.timestamp.data  
 sign = self.sign.data  
 print('客户端提交的sign: ', sign)

# md5(timestamp+telephone+salt) 加密  
 # md5函数必须传递一个bytes类型的字符串进去  
 sign2 = hashlib.md5(（timestamp + telephone + self.salt).encode('utf-8')） .hexdigest()

# hexdigest函数返回摘要，作为十六进制数据字符串值

print('服务器生成的sign: ', sign2)  
 if sign == sign2:  
 return True  
 else:  
 return False

1. 在公共视图文件apps/common/views.py中重写短信验证码发送视图函数，把原来的注释掉

# 短信验证码发送视图函数 加密功能改进版  
@bp.route('/sms\_captcha/', methods=['POST'])  
def sms\_captcha():  
 # telephone  
 # timestamp  
 # md5(ts+telephone+salt)  
 form = SMSCaptchaForm(request.form)  
 if form.validate():  
 telephone = form.telephone.data  
 captcha = Captcha.gene\_text(number=6)  
 if alidayu.send\_sms(telephone, code=captcha):  
 return restful.success()  
 else:  
 # return restful.params\_error(message='短信验证码发送失败！')  
 return restful.success() # 为了方便开发，假设短信发送成功  
 else:  
 return restful.params\_error(message='参数错误！')

1. 加载MD5还有csrf\_token()到前台注册页面模板front\_signup.html中的head标签中

<head>

…

<meta name="csrf-token" content="{{ csrf\_token() }}">

…

<script src="https://cdn.bootcss.com/blueimp-md5/2.10.0/js/md5.min.js"></script>

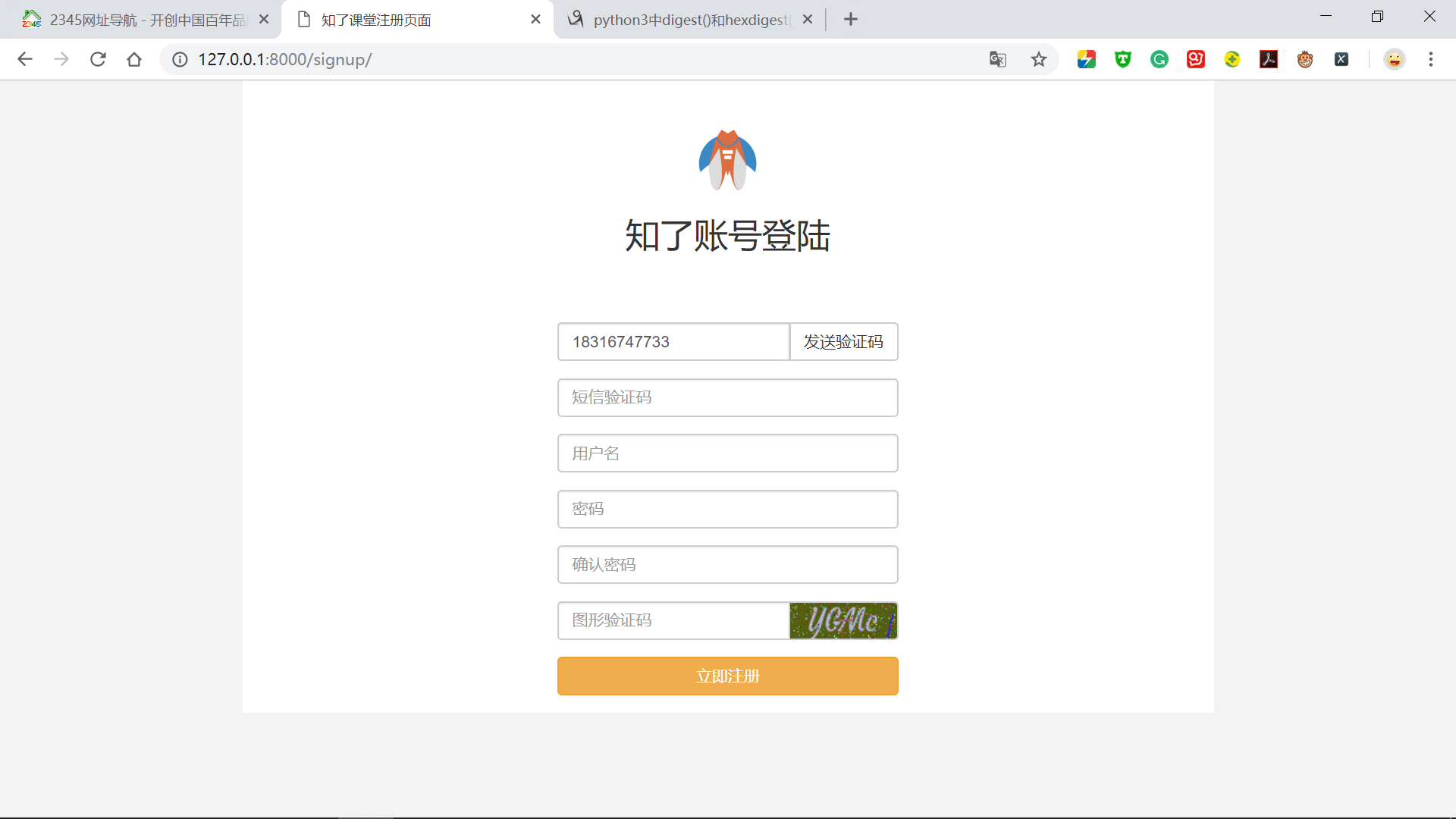
…

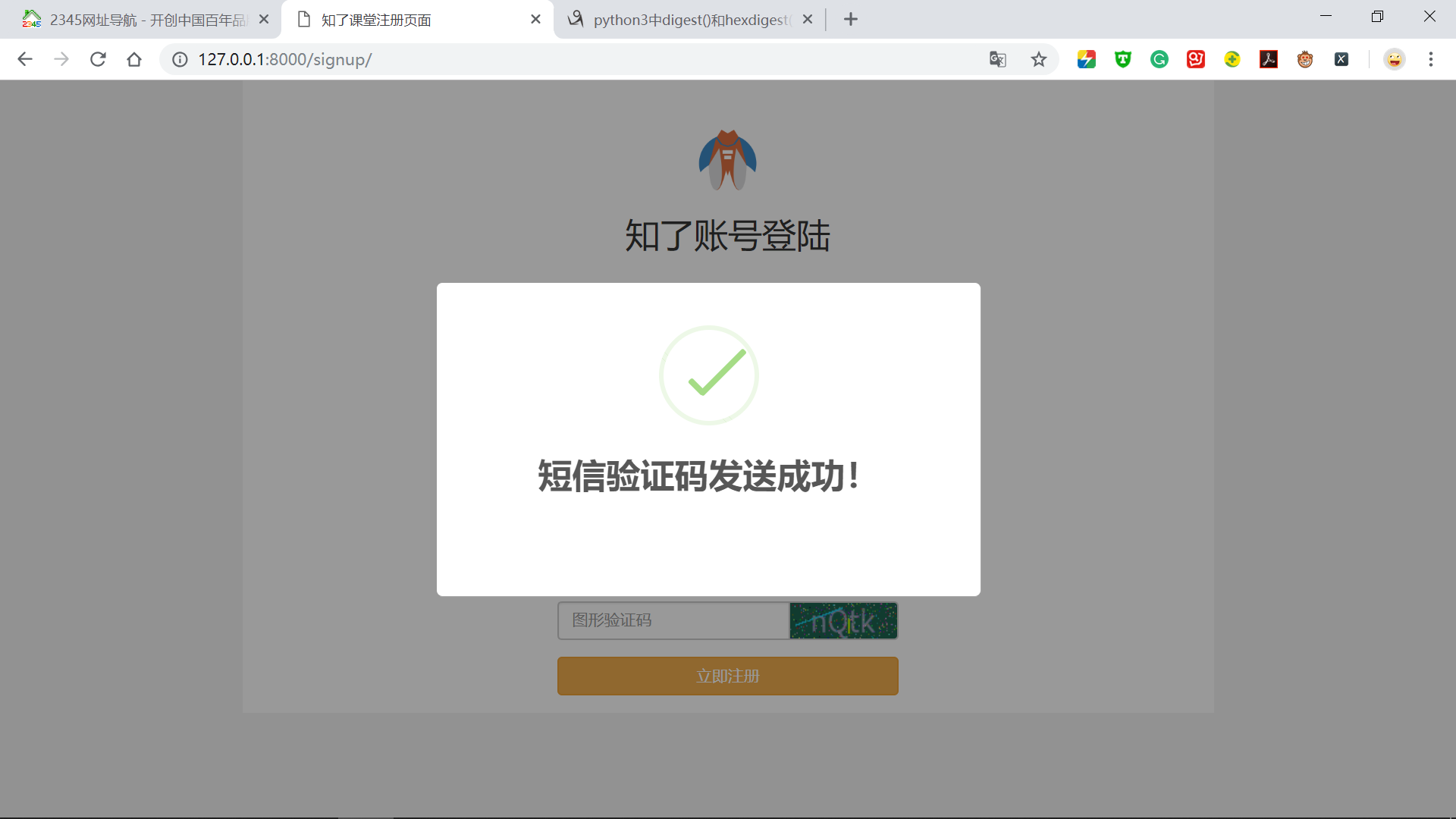
<head>

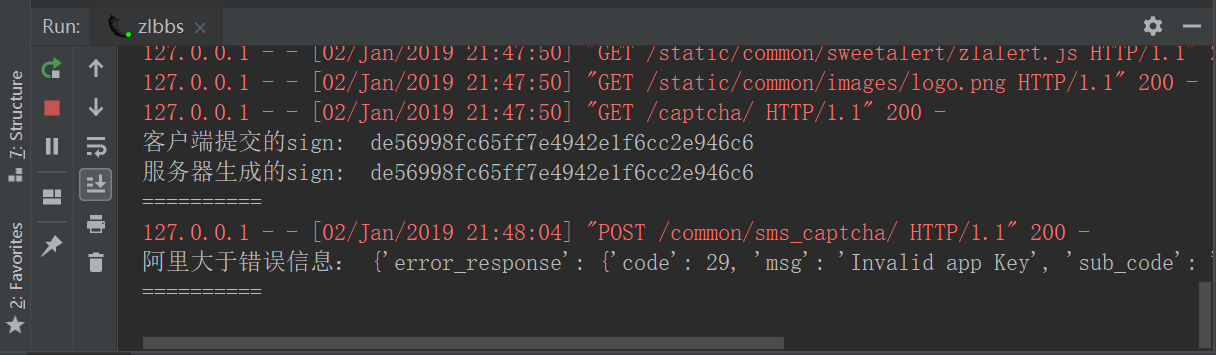
1. 修改static/front/js/front\_signup.js：

// 短信验证码点击发送  
$(**function** () {  
 $('#sms-captcha-btn').click(**function** (event) {  
 event.preventDefault();  
  
 **var** self = $(**this**);  
 **var** telephone = $("input[name='telephone']").val();  
  
 **if** (!(/^1[345789]\d{9}$/.test(telephone))) {  
 zlalert.alertInfoToast('请输入正确的手机号码！');  
 **return**;  
 }  
  
 **var** timestamp = (**new** Date).getTime(); 修改  
 **var** sign = md5(timestamp + telephone + 'hihasiyfbfwe&($^\*&hunggfaM'); 修改  
  
 zlajax.post({ 这里改成post方法  
 'url': '/common/sms\_captcha/', 修改  
 'data': { 修改  
 'telephone': telephone,  
 'timestamp': timestamp,  
 'sign': sign  
 },  
 'success': **function** (data) {  
 **if** (data['code'] == 200) {  
 zlalert.alertSuccessToast('短信验证码发送成功！');  
 self.attr('disable', 'disabled');  
 **var** timeCount = 60; // 验证码有效期60秒  
 **var** timer = setInterval(**function** () { 修改  
 timeCount--;  
 self.text(timeCount);  
 **if** (timeCount <= 0) { 修改  
 self.removeAttr('disabled');  
 clearInterval(timer);  
 self.text('发送验证码')  
 }  
 }, 1000);  
 } **else** {  
 zlalert.alertInfoToast(data['message']);  
 }  
 }  
 });  
 });  
});

1. 启动服务，打开链接<http://127.0.0.1:8000/signup/>，输入手机号码，点击发送验证码。。。







# 缓存验证码

1. 短信验证码缓存，修改apps/common/views.py视图文件

from utils import restful, zlcache

…

# 短信验证码发送视图函数 加密功能改进版  
@bp.route('/sms\_captcha/', methods=['POST'])  
def sms\_captcha():  
 # telephone  
 # timestamp  
 # md5(ts+telephone+salt)  
 form = SMSCaptchaForm(request.form)  
 if form.validate():  
 telephone = form.telephone.data  
 captcha = Captcha.gene\_text(number=6)  
 if alidayu.send\_sms(telephone, code=captcha):  
 zlcache.set(telephone, captcha) # 将验证码缓存到memcached系统中  
 return restful.success()  
 else:  
 # return restful.params\_error(message='短信验证码发送失败！')  
 zlcache.set(telephone, captcha) # 这里是为了方便开发工作  
 return restful.success() # 为了方便开发，假设短信发送成功  
 else:  
 return restful.params\_error(message='参数错误！')

1. 图形验证码缓存

先将图形验证码生成视图函数从apps/front/views.py剪切到apps/common/views.py中，

apps/common/views.py添加的代码，然后添加memcached缓存：

from flask import Blueprint, request, make\_response  
from exts import alidayu  
from utils import restful, zlcache  
from utils.captcha import Captcha  
from apps.common.forms import SMSCaptchaForm  
from io import BytesIO

…

# 图形验证码生成视图函数  
@bp.route('/captcha/')  
def graph\_captcha():  
 text, image = Captcha.gene\_graph\_captcha()

zlcache.set(text.lower(), text.lower()) # 把验证码添加到memcached缓存中  
 out = BytesIO()  
 image.save(out, 'png')  
 out.seek(0)  
 resp = make\_response(out.read())  
 resp.content\_type = 'image/png'  
 return resp

修改templates/front/front\_signup.html中图形验证码的视图路径：

<div class="form-group">  
 <div class="input-group">  
 <input type="text" class="form-control" name="graph\_captcha" placeholder="图形验证码">  
 <span class="input-group-addon captcha-addon">  
 <img id="captcha-img" class="captcha-img" src="{{ url\_for('common.graph\_captcha') }}" alt="">  
 </span>  
 </div>  
</div>

# front前台注册功能完成

1. 注册功能后端逻辑代码完成

在前台apps/front/forms.py中定义注册验证表单类：

from ..forms import BaseForm  
from wtforms import StringField  
from wtforms.validators import Regexp, EqualTo, ValidationError  
from utils import zlcache  
  
class SignupForm(BaseForm):  
 telephone = StringField(validators=[Regexp(r'1[345789]\d{9}', message='请输入正确格式的手机号码！')])  
 sms\_captcha = StringField(validators=[Regexp(r'\w{6}', message='请输入正确格式的短信验证码！')])  
 username = StringField(validators=[Regexp(r'.{2,20}', message='请输入正确格式的用户名！')])  
 password1 = StringField(validators=[Regexp(r'[0-9a-zA-Z\_\.]{6,20}', message='请输入正确格式的密码！')])  
 password2 = StringField(validators=[EqualTo('password1', message='两次输入的密码不一致！')])  
 graph\_captcha = StringField(validators=[Regexp(r'\w{4}', message='请输入正确格式的短信验证码！')])  
  
 def validate\_sms\_captcha(self, field):  
 sms\_captcha = field.data  
 telephone = self.telephone.data  
  
 sms\_captcha\_mem = zlcache.get(telephone)  
 if not sms\_captcha\_mem or sms\_captcha\_mem.lower() != sms\_captcha.lower():  
 raise ValidationError(message='短信验证码错误！')  
  
 def validate\_graph\_captcha(self, field):  
 graph\_captcha = field.data  
  
 graph\_captcha\_mem = zlcache.get(graph\_captcha.lower())  
 if not graph\_captcha\_mem:  
 raise ValidationError(message='图形验证码错误！')

1. 注册功能前端逻辑代码完成

在前台注册页面模板文件templetes/front/front\_signup.html中给立即注册按钮添加一个id：

<div class="form-group">  
 <button id="submit-btn" class="btn btn-warning btn-block">立即注册</button>  
</div>

在前端文件static/front/js/front\_signup.js中实现注册功能：

// 前台注册功能  
$(**function** () {  
 $('#submit-btn').click(**function** (event) {  
 event.preventDefault();  
  
 **var** telephone\_input = $("input[name='telephone']");  
 **var** sms\_captcha\_input = $("input[name='sms\_captcha']");  
 **var** username\_input = $("input[name='username']");  
 **var** password1\_input = $("input[name='password1']");  
 **var** password2\_input = $("input[name='password2']");  
 **var** graph\_captcha\_input = $("input[name='graph\_captcha']");  
  
 **var** telephone = telephone\_input.val();  
 **var** sms\_captcha = sms\_captcha\_input.val();  
 **var** username = username\_input.val();  
 **var** password1 = password1\_input.val();  
 **var** password2 = password2\_input.val();  
 **var** graph\_captcha = graph\_captcha\_input.val();  
  
 zlajax.post({  
 'url': '/signup/',  
 'data': {  
 'telephone': telephone,  
 'sms\_captcha': sms\_captcha,  
 'username': usename,  
 'password1': password1,  
 'passqord2': password2,  
 'graph\_captcha': graph\_captcha  
 },  
 'success': **function** (data) {  
 **if** (data['code'] == 200) {  
 window.location = '/';  
 } **else** {  
 zlalert.alertInfo(data['message']);  
 }  
 },  
 'fail': **function** (error) {  
 zlalert.alertNetworkError();  
 }  
 });  
 });  
});

1. 在前台视图文件apps/front/views.py的注册类视图中实现注册表单验证和添加新的前台用户

from flask import (  
 …  
 request,  
)  
from .forms import SignupForm  
from utils import restful  
from .models import FrontUser  
from exts import db

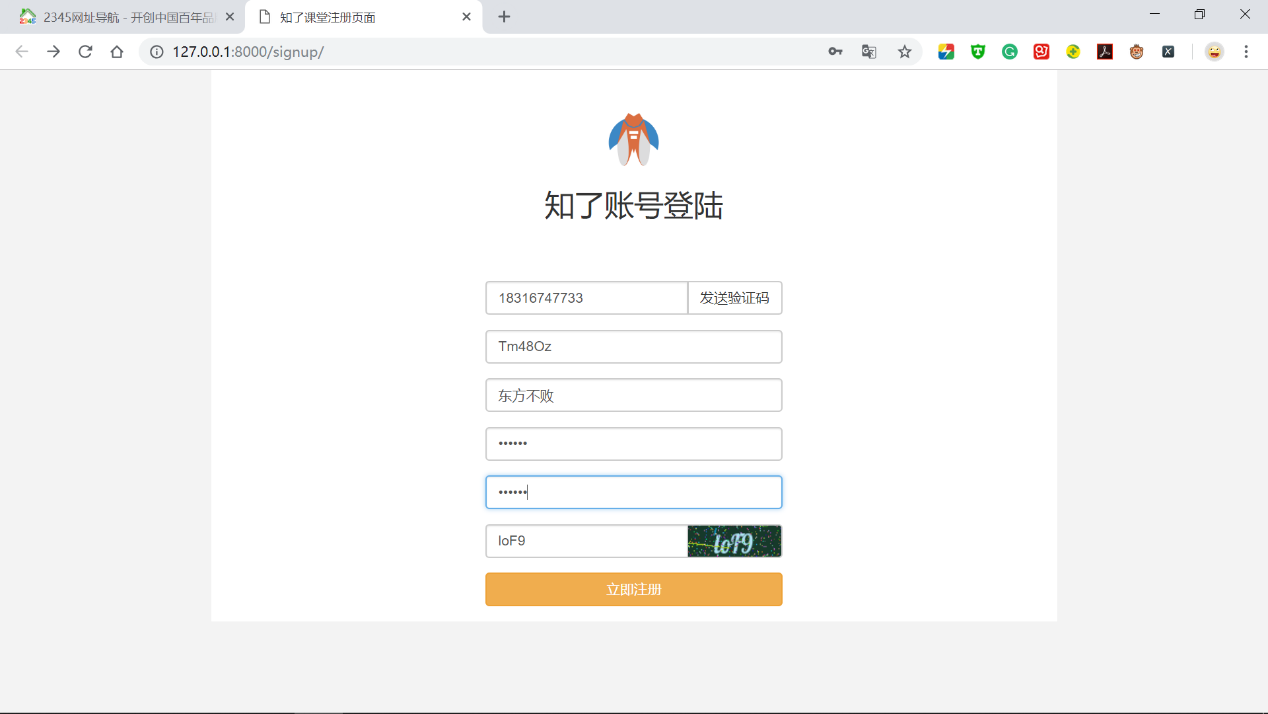
…

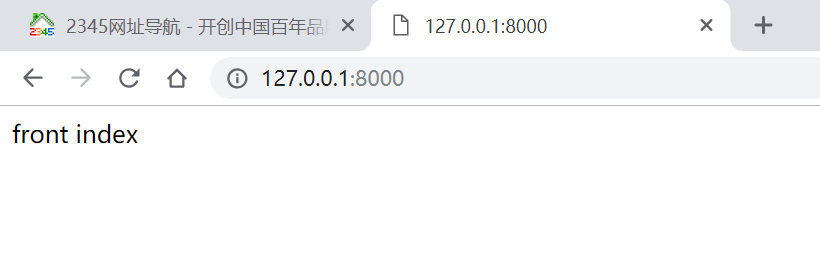
class SignupView(views.MethodView):  
 def get(self):  
 return render\_template('front/front\_signup.html')  
  
 def post(self):  
 # 验证表单  
 form = SignupForm(request.form)  
 if form.validate():  
 telephone = form.telephone.data  
 username = form.username.data  
 password = form.password1.data  
  
 # 创建新的前台用户  
 user = FrontUser(telephone=telephone, username=username, password=password)  
 db.session.add(user)  
 db.session.commit()  
 return restful.success()  
 else:  
 print(form.get\_error())  
 return restful.params\_error(message=form.get\_error())

1. 在公共视图文件common/views.py的短信验证码发送视图函数中打印出短信验证码，方便开发工作

# 短信验证码发送视图函数 加密功能改进版  
@bp.route('/sms\_captcha/', methods=['POST'])  
def sms\_captcha():  
 # telephone  
 # timestamp  
 # md5(ts+telephone+salt)  
 form = SMSCaptchaForm(request.form)  
 if form.validate():  
 telephone = form.telephone.data  
 captcha = Captcha.gene\_text(number=6)  
 print('发送的短信验证码是：', captcha) # 打印出短信验证码  
 if alidayu.send\_sms(telephone, code=captcha):  
 zlcache.set(telephone, captcha) # 将验证码缓存到memcached系统中  
 return restful.success()  
 else:  
 # return restful.params\_error(message='短信验证码发送失败！')  
 zlcache.set(telephone, captcha) # 这里是为了方便开发工作  
 return restful.success() # 为了方便开发，假设短信发送成功  
 else:  
 return restful.params\_error(message='参数错误！')

1. 启动服务，输入手机号码，获取短信验证码，然后输入用户信息，点击立即注册按钮，就会跳转到前台主页面，数据库表front\_user也会保存了一个用户信息。。。







# front前台注册完成跳转到上一个页面

# 。。。